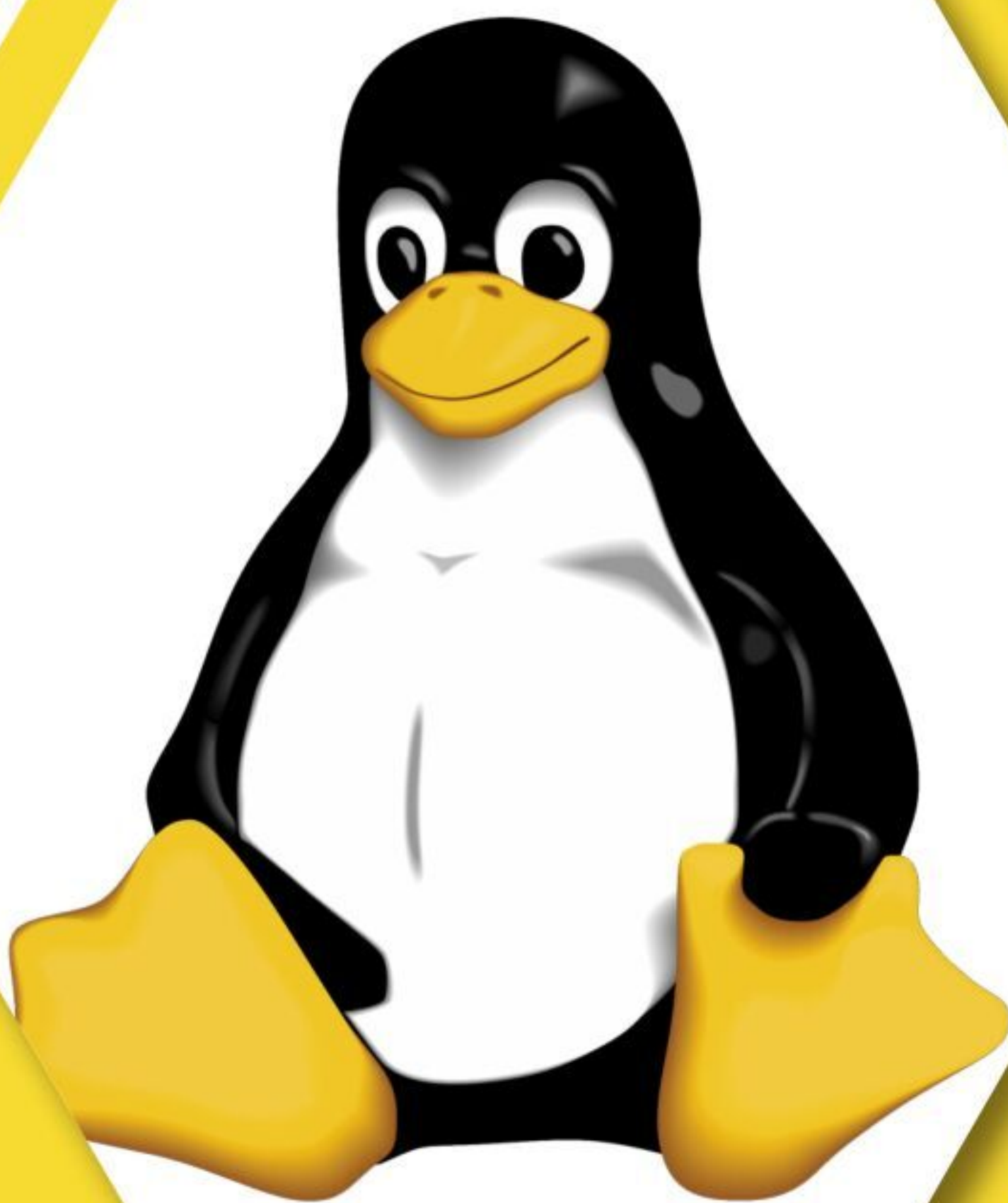


NEW

Coding For Beginners

All you need to get started
with Coding & Programming

Over
430
Tips & Hints
Inside



- ✓ Jargon-free Tips & Advice
- ✓ Step-by-step Tutorials
- ✓ Clear Full Colour Guides



Coding For Beginners

Starting something new can be daunting. Learning a skill or mastering a new piece of hardware is tough. Even tougher if you have no-one at hand to help. Conversely as the complexity of our consumer technology increases, the size of the requisite instruction manual decreases or in some cases it simply disappears. At numerous times in our lives we have all been “beginners”, there is no shame in that fact and rightly so. How many times have you asked aloud, “What does this button do?”. “Why doesn’t that work?”. “What do you mean it doesn’t do that?”. “HELP!”. At the start of any new journey or adventure we are all beginners but fortunately for you we are here to stand beside you at every stage.

Over this extensive series of titles we will be looking in great depth at the latest consumer electronics, software, hobbies and trends out of the box! We will guide you step-by-step through using all aspects of the technology that you may have been previously apprehensive at attempting. Let our expert guide help you build your technology understanding and skills, taking you from a novice to a confident and experienced user.

Over the page our journey begins. We would wish you luck but we’re sure with our support you won’t need it.





Welcome, Future Coder

Everything you do online, every time you power on your smart TV, whenever you use your in-car GPS, use your phone, play a game on a tablet, console or PC, it's all been coded by a group of people. All these ones and zeros are developed by those who have learned how to code, and with this book you too can learn how to get started on the road to becoming a programmer.

We cover C++, Python, and Linux Scripting within these pages. There's a huge section of project ideas, type-in listings, and in-depth looks at how code works. But first, let's begin with a good foundation.



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.

**“It’s up to you
how far to take
your coding
adventure!”**

```

1 #include<stdio.h>
2 #include<dos.h>
3 #include<stdlib.h>
4 #include<conio.h>
5 void setup()
6 {
7     textcolor(BLACK);
8     textbackground(15);
9     clrscr();
10    window(10,2,70,3);
11    printf("Press X to Exit, Press Space to Jump");
12    window(62,2,80,3);
13    printf("SCORE : ");
14    window(1,25,80,25);
15    for(int x=0;x<79;x++)
16        printf(" ");
17    textcolor(0);
18 }
19
20 int t,speed=40;
21 void ds(int jump=0)
22 {
23     static int a=1;
24
25     if(jump==0)
26         t=0;
27     else if(jump==2)
28         t--;
29     else t++;
30     window(2,15-t,18,25);
31     printf(" ");
32     printf("      ММММММММ");
33     printf("      ММММММММ");
34     printf("      ММММММММ");
35     printf("      ММММММММ");
36     printf("      ММММММММММММ");
37     printf("      ММММММММММММММММ");
38     printf("      ММММММММММММ");
39     if(jump==1 || jump==2){
40         printf("      ММММММММ");
41         printf("      ММММММММ");
42     }else if(a==1)
43     {
44         printf("      ММММММММММММ");
45         printf("      ММММММММММММ");
46         a=2;
47     }
48     else if(a==2)
49     {
50         printf("      ММММММММММММ");
51         printf("      ММММММММММММ");
52         a=1;
53     }
54     printf(" ");
55     delay(speed);
56 }
57 void obj()
58 {

```

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

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!



A Brief History of Coding

It's easy to think that programming a machine to automate a process or calculate a value is a modern concept that's only really happened in the last fifty years or so. However, that assumption is quite wrong, coding has actually been around for quite some time.

01000011 01101111 01100100 01100101

Essentially all forms of coding are made up of ones and zeros, on or off states. This works for a modern computer and even the oldest known computational device.

~87 BC

~850 AD

1800

1842-1843

1930-1950

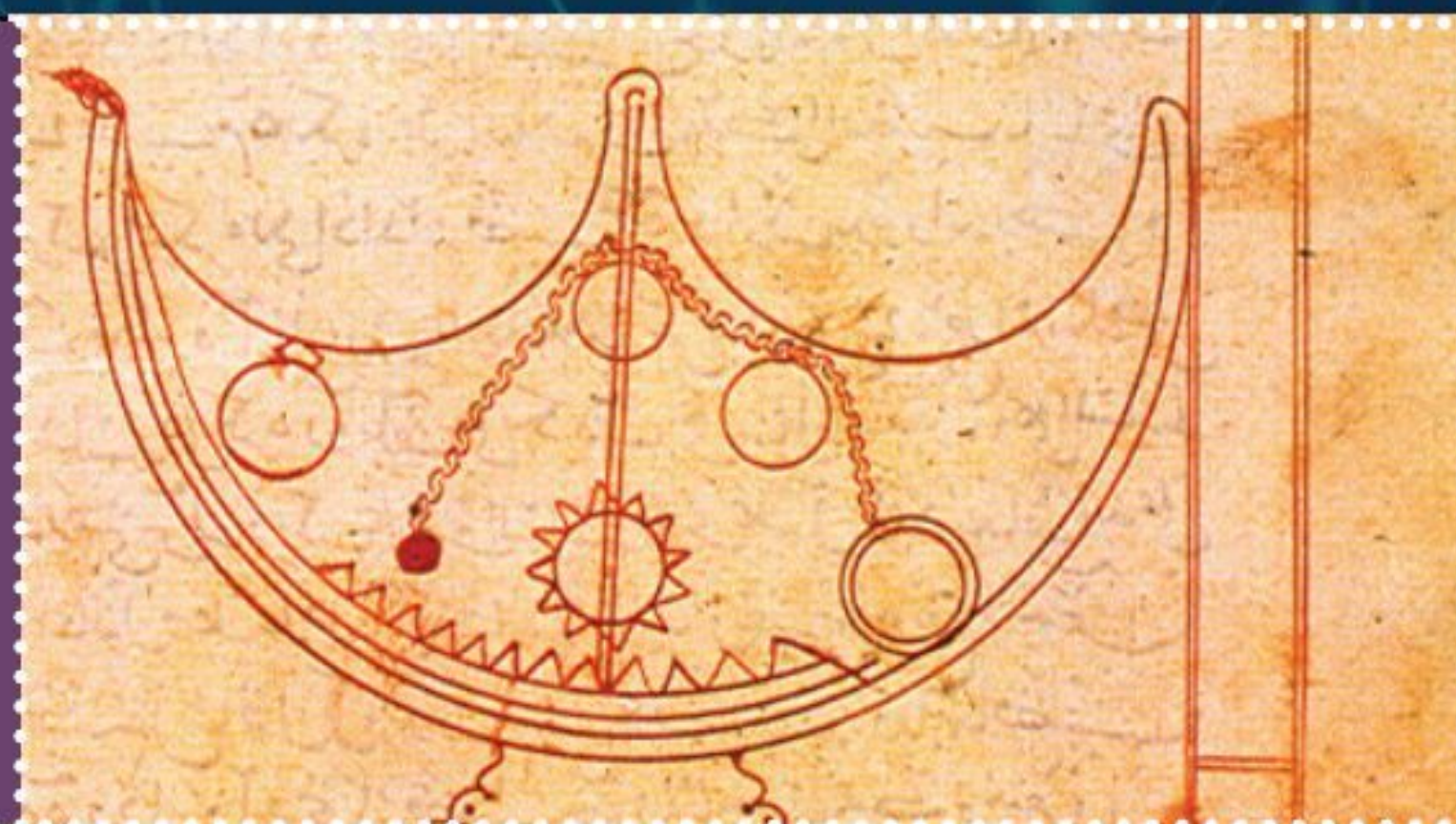
It's difficult to pinpoint an exact start of when humans began to 'program' a device. However, it's widely accepted that the Antikythera Mechanism is possibly the first 'coded' artefact. It's dated to about 87 BC and is an ancient Greek analogue computer and orrery used to predict astronomical positions.



Joseph Marie Jacquard invents a programmable loom, which used cards with punched holes to create the textile design. However, it is thought that he based his design on a previous automated weaving process from 1725, by Basile Bouchon.



The Banū Mūsā brothers, three Persian scholars who worked in the House of Wisdom in Baghdad, published the Book of Ingenious Devices in around 850 AD. Among the inventions listed was a mechanical musical instrument, a hydro-powered organ that played interchangeable cylinders automatically.



Ada Lovelace translated the memoirs of the Italian mathematician, Francis Manegrand, regarding Charles Babbage's Analytical Engine. She made copious notes within her writing, detailing a method of calculating Bernoulli Numbers using the engine. This is recognised as the first computer program. Not bad, considering there were no computers available at the time.

During the Second World War, there significant advances were made in programmable machines. Most notably, the cryptographic machines used to decipher military codes used by the Nazis. The Enigma was invented by the German engineer Arthur Scherbius but was made famous by Alan Turing at Bletchley Park's codebreaking centre.



From the 1970s, the development of the likes of C, SQL, C with Classes (C++), MATLAB, Common Lisp and more came to the fore. The '80s was undoubtedly the golden age of the home computer, a time when silicon processors were cheap enough for ordinary folk to buy. This led to a boom in home/bedroom coders with the rise of 8-bit machines.



1951-1958

1959

1960-1970

1970-1985

1990s-Present Day

```

MONITOR FOR 6802 1.4      9-14-80  TSC ASSEMBLER  PAGE  2

C000 8E 00 70  START  ORG  ROM+$0000 BEGIN MONITOR
                        LDS  #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013  RESETA EQU 40010011
0011  CTLREG EQU 40010001

C003 86 13  INITA  LDA A #RESETA  RESET ACIA
C005 B7 80 04  STA A ACIA
C008 86 11  LDA A #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04  STA A ACIA
C00D 7E C0 F1  JMP  SIGNON  GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* CALLS: none
* DESTROYS: acc A
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH  LDA A ACIA  GET STATUS
C013 47  ASR A  SHIFT RORF FLAG INTO CARRY
C014 24 FA  BCC  INCH  RECEIVE NOT READY
C016 B6 80 05  LDA A ACIA+1  GET CHAR
C019 84 7F  AND A #7F  MASK PARITY
C01B 7E C0 79  JMP  OUTCH  ECHO & RTS

*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0  INHEX  BSR  INCH  GET A CHAR
C020 81 30  CMP A #'0  ZERO
C022 2B 11  BMI  HEXERR  NOT HEX
C024 81 39  CMP A #'9  NINE
C026 2F 0A  BLE  HEXRTS  GOOD HEX
C028 81 41  CMP A #'A  NOT HEX
C02A 2B 09  BMI  HEXERR  NOT HEX
C02C 81 46  CMP A #'F  NOT HEX
C02E 2E 05  BGT  HEXERR
C030 80 07  SUB A #7  FIX A-F
C032 84 0F  HEXRTS AND A #50F  CONVERT ASCII TO DIGIT
C034 39  RTS
C035 7E C0 AF  HEXERR JMP  CTRL  RETURN TO CONTROL LOOP

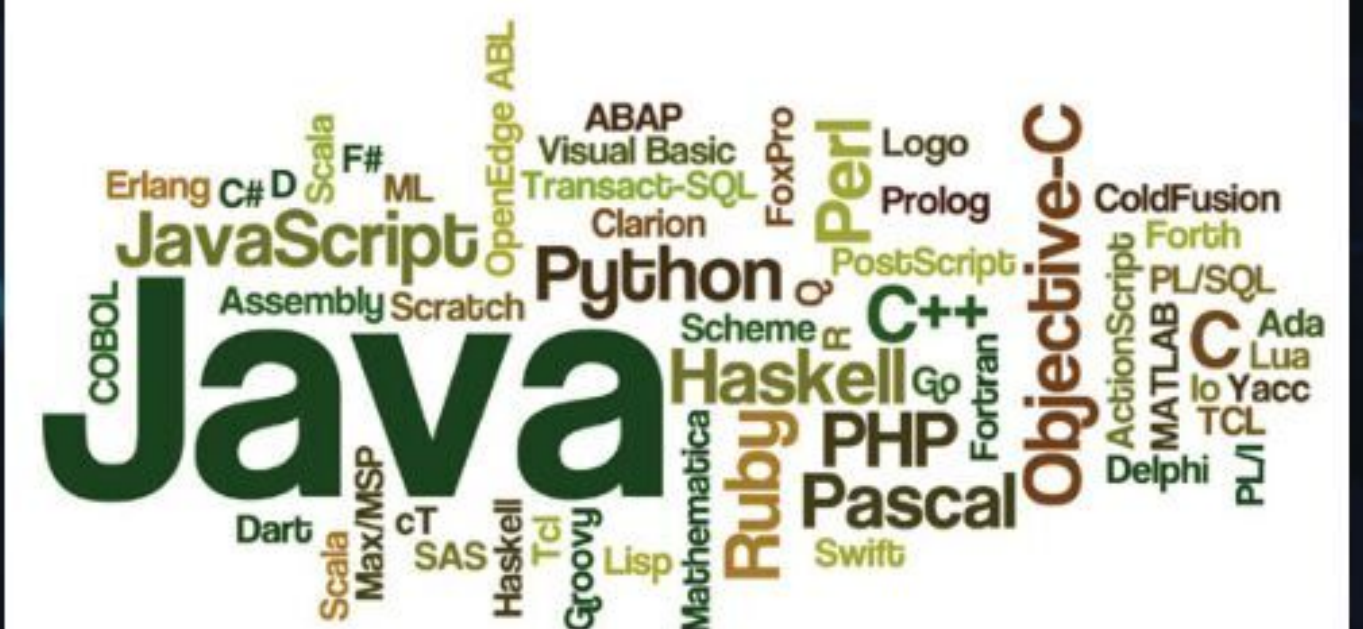
```

Computer programming was mainly utilised by universities, the military and big corporations during the '60s and the '70s. A notable step toward a more user-friendly, or home user language, was the development of BASIC (Beginners All-purpose Symbolic Instruction Code) in the mid-sixties.

```

10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
25 REM
30 INPUT "How many stars do you want: "; N
35 S$ = ""
40 FOR I = 1 TO N
50 S$ = S$ + "*"
55 NEXT I
60 PRINT S$
65 REM
70 INPUT "Do you want more stars? "; A$
80 IF LEN(A$) = 0 THEN GOTO 70
90 A$ = LEFT$(A$, 1)
100 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 30
110 PRINT "Goodbye ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT

```



The Internet age brought a wealth of new programming languages and allowed people access to the tools and knowledge needed to learn coding in a better way. Not only could a user learn how to code but they could freely share their code and source other code to improve their own.

The first true computer code was Assembly Language (ASM) or Regional Assembly Language. ASM was specific to the architecture of the machine it was being used on. In 1951 programming languages fell under the generic term Autocode. Soon languages such as IPL, FORTRAN and ALGOL 58 were developed.

Admiral Grace Hopper was part of the team that developed the UNIVAC I computer and she eventually developed a compiler for it. In time, the compiler she developed became COBOL (Common Business-oriented Language), a computer language that's still in use today.





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

ph



python



**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 for the iPad.



Creating a Coding Platform

The term 'Coding Platform' can signify: a type of hardware on which you can code, a particular operating system, or even a custom environment that's pre-built and designed for easy game creation. In truth it's quite a loose term, as a Coding Platform can be a mixture of all of these ingredients, it's simply down to which programming language you intend to code in and what your end goals are.

Coding can be one of those experiences that sounds fantastic, but is often confusing to tackle. After all, there's a plethora of languages to choose from, countless apps that will enable you to code in a specific, or range, of languages and an equally huge amount of third-party software to consider. In addition, by accessing the Internet, you will discover that there are countless coding tutorials available for the language in which you've decided you want to program, alongside even more examples of code. It's all a little too much at first.

The trick is to slow down and, to begin with, not look too deeply into coding. Like all good projects, you need a solid foundation on which to build your skillset and to have all the necessary tools available to enable you to complete the basic steps. This is where creating a coding platform comes in, as it will be your learning foundation while you begin to take your first tentative steps into the wider world of coding.

```
5
6 from django.contrib.auth.decorators import login_required, permission_required
7 from django.contrib.admin.views.decorators import staff_member_required
8 from django.core.exceptions import ObjectDoesNotExist
9 from django.http import HttpResponseForbidden
10 from django.contrib.auth import authenticate, login, logout
11 from django.contrib.auth.models import User
12
13 from django.core.paginator import Paginator
14 from django.core.urlresolvers import reverse
15 from django.conf import settings
16 from django.db.models import Q, Sum, Min, Count
17
18 from django.contrib import messages
19 from datetime import date, timedelta
20 import re
21
22 from .models import *
23
24
25 ERROR_MESSAGE = "There are some errors, please check again"
```

HARDWARE

Thankfully, coding at the foundation level doesn't require specialist equipment, or a top of the range, liquid hydrogen-cooled PC. If

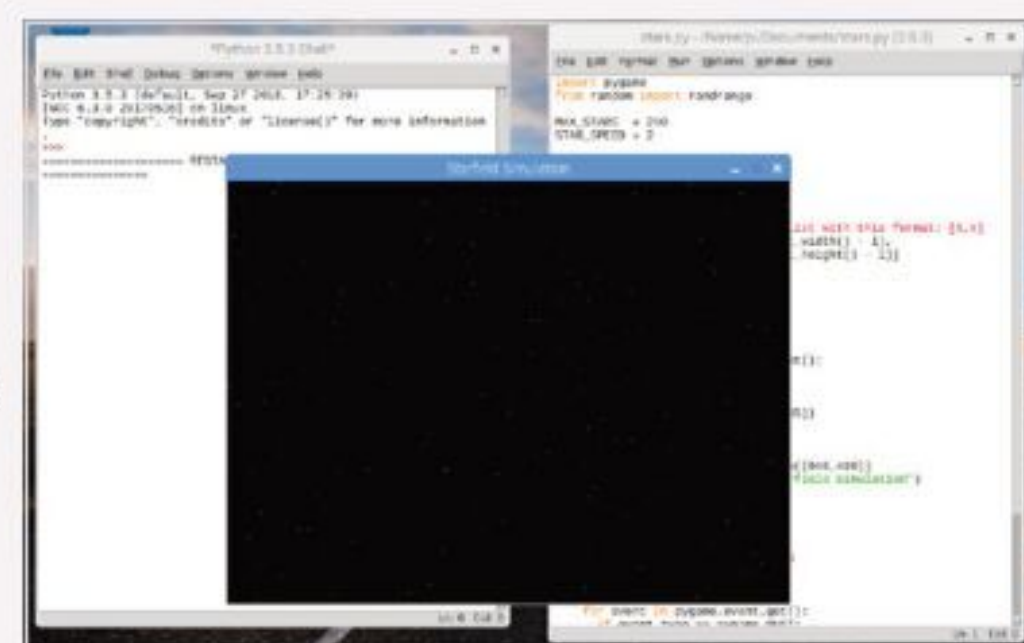


you own a computer, no matter how basic, you can begin to learn how to code. Naturally, if the computer in question is a Commodore 64 then you may have some difficulty following a modern language tutorial, but some of the best programmers around today started on an 8-bit machine, so there's hope yet.

You will need access to the Internet to download, install and update the coding development environment, alongside a computer with Windows 10, macOS, or Linux, installed. You can use other operating systems, but these are the 'big three' and you will find that most code resources are written with one, or all, of these in mind.

SOFTWARE

In terms of software, most of the development environments have the tools that allow you to code, compile the code and execute it, freely available to download and install. There are some specialist tools available that will cost, but at this level they're not necessary, so don't be fooled into thinking you need to purchase any extra software in order to start learning how to code.



Over time, you may find yourself progressing from the mainstream development environment and using a collection of your own, discovered, tools to write your code. It's all personal preference in the end and as you become more experienced, you will start to use different tools to get the job done.

OPERATING SYSTEMS

Windows 10 is the most widely used operating system in the world, so it's natural that the vast majority of coding tools are written for Microsoft's leading operating system. However, don't discount macOS and especially Linux.

macOS users enjoy an equal number of coding tools to their Windows counterparts. In fact, you will probably find that a lot of professional coders use a Mac over a PC, simply because of the fact that the Mac operating system is built on top of Unix (the command-line OS that powers much of the world's filesystems and servers). This Unix layer lets you test programs in almost any language without using a specialised IDE.

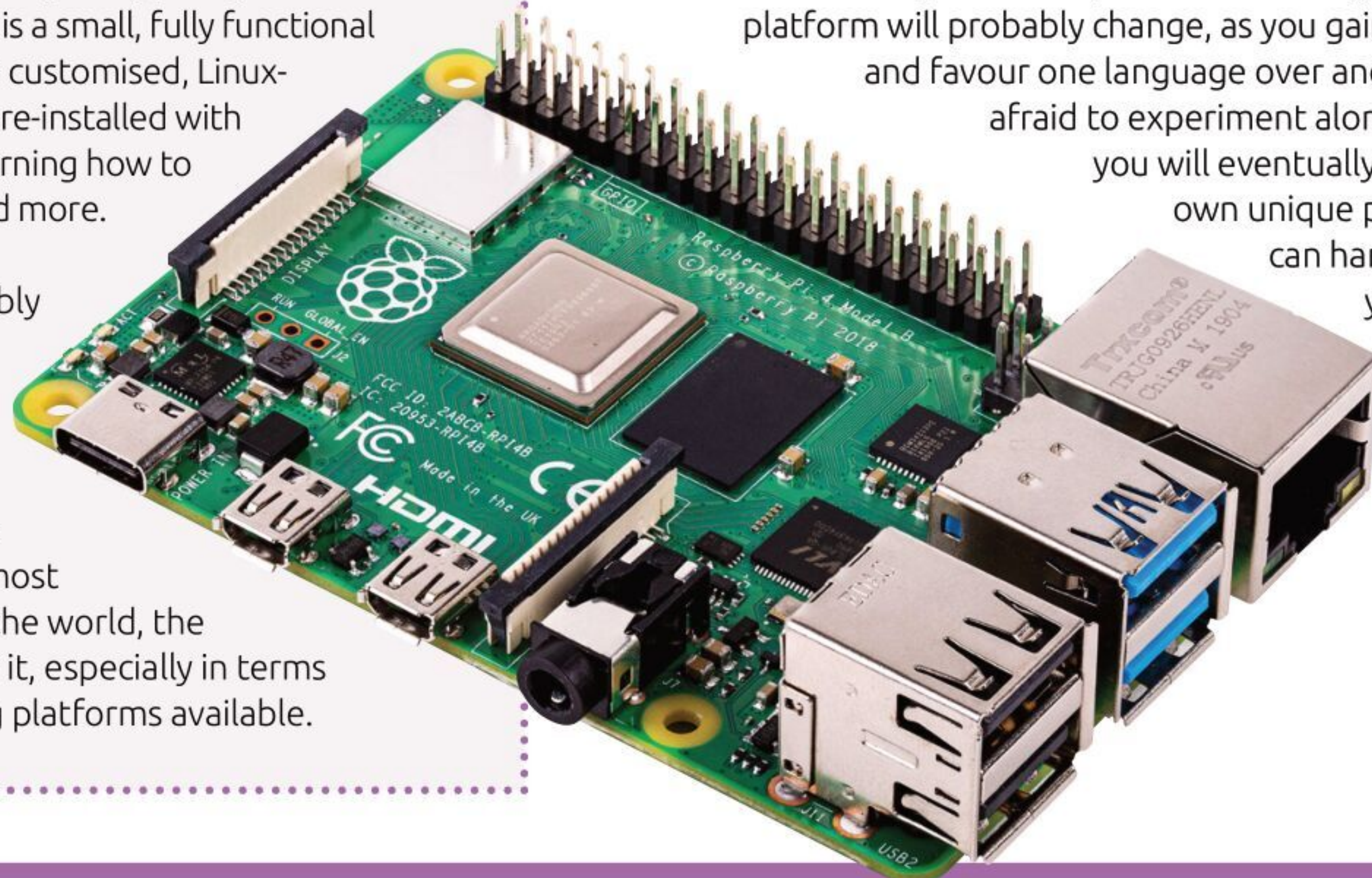


However, Linux is, by far and away, one of the most popular and important coding operating systems available. Not only does it have a Unix-like backbone, it's also free to download, install and use and comes with most of the tools necessary to start learning how to code. Linux powers most of the servers that make up the Internet. It's used on nearly all of the top supercomputers, as well as specifically in organisations such as NASA, CERN and the military, it also forms the base of Android-powered devices, smart TVs and in-car systems. Linux, as a coding platform, is an excellent idea and it can be installed inside a virtual machine without ever affecting the installation of Windows or macOS.

THE RASPBERRY PI

If you haven't already heard of the Raspberry Pi, then we suggest you head over to www.raspberrypi.org and check it out. In short, the Raspberry Pi is a small, fully functional computer. It comes with its own customised, Linux-based operating system that's pre-installed with everything you need to start learning how to code in Python, C++, Scratch and more.

Costing around £35, it's incredibly cheap and allows you to utilise different hardware, in the form of robotics and electronics projects, as well as offering a complete desktop experience. Although not the most powerful computing device in the world, the Raspberry Pi has a lot going for it, especially in terms of being one of the best coding platforms available.



VIRTUAL MACHINES

A virtual machine is a piece of software that allows you to install a fully working operating system within the confines of the software itself. The installed OS will allocate user-defined resources from the host computer, providing memory, hard drive space etc., as well as sharing the host computer's Internet connection.

The advantage of a virtual machine is that you can work with Linux, for example, without it affecting your currently installed host OS. This means that you can have Windows 10 running and launch your virtual machine client, boot into Linux and use all the functionality of Linux, while still being able to use Windows.

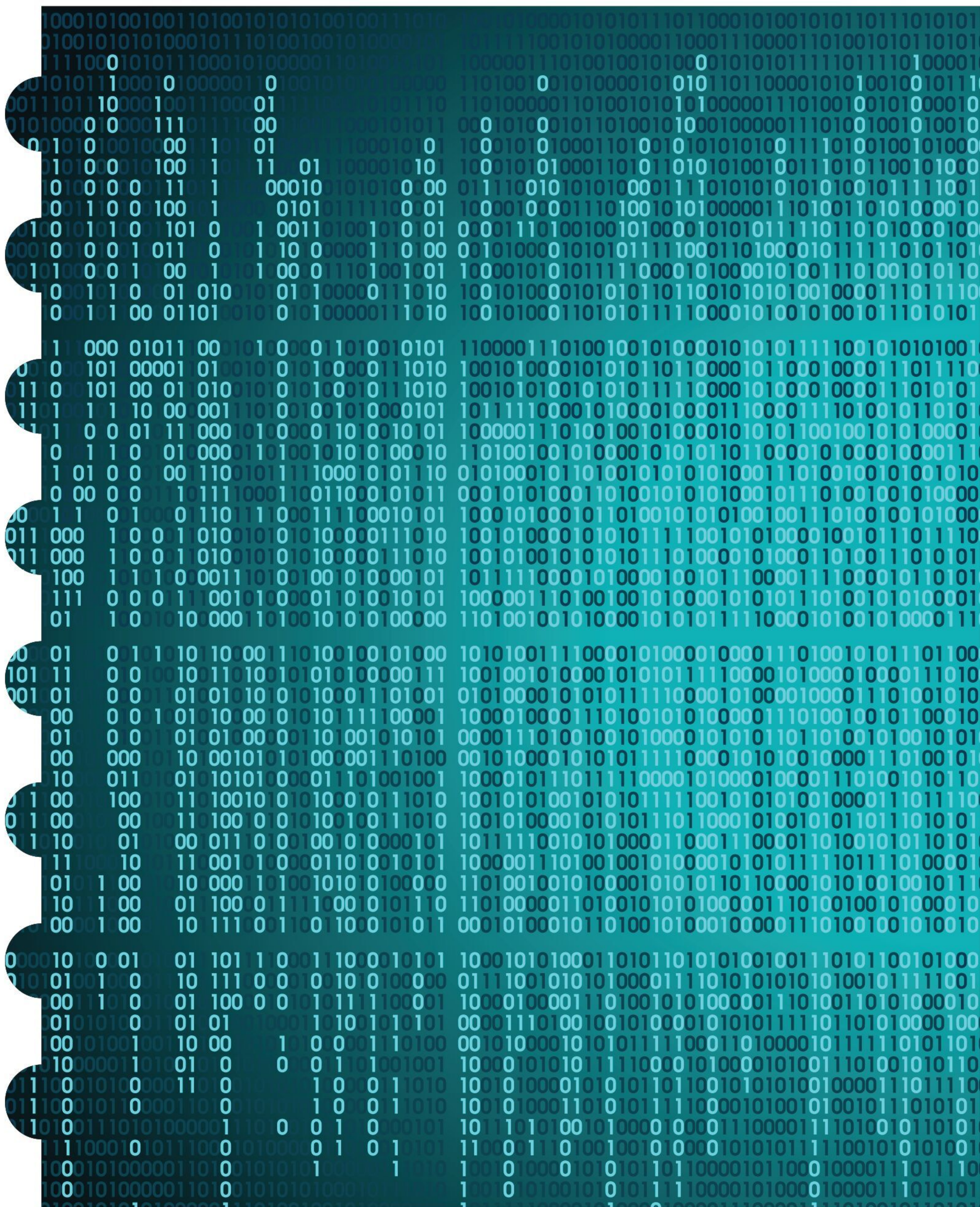


This, of course, makes it a fantastic coding platform, as you can have different installations of operating systems running from the host computer while using different coding languages. You can test your code without fear of breaking your host OS and it's easy to return to a previous configuration without the need to reinstall everything again.

Virtualisation is the key to most big companies now. You will probably find, rather than having a single server with an installation of Windows Server, for example, the IT team have instead opted for a virtualised environment whereby each Windows Server instance is a virtual machine running from several powerful machines. This cuts down on the number of physical machines, allows the team to better manage resources and enables them to deploy an entire server dedicated to a particular task in a fraction of the time.

YOUR OWN CODING PLATFORM

Whichever method you choose, remember that your coding platform will probably change, as you gain experience and favour one language over another. Don't be afraid to experiment along the way, as you will eventually create your own unique platform that can handle all the code you enter into it.



Introducing C++

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++ will open 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, and much more.



Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed to 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 both the complexity of the language 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 Ph.D. thesis. Stroustrup's plan was to further the original C language, which had been widely used since the early seventies.

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

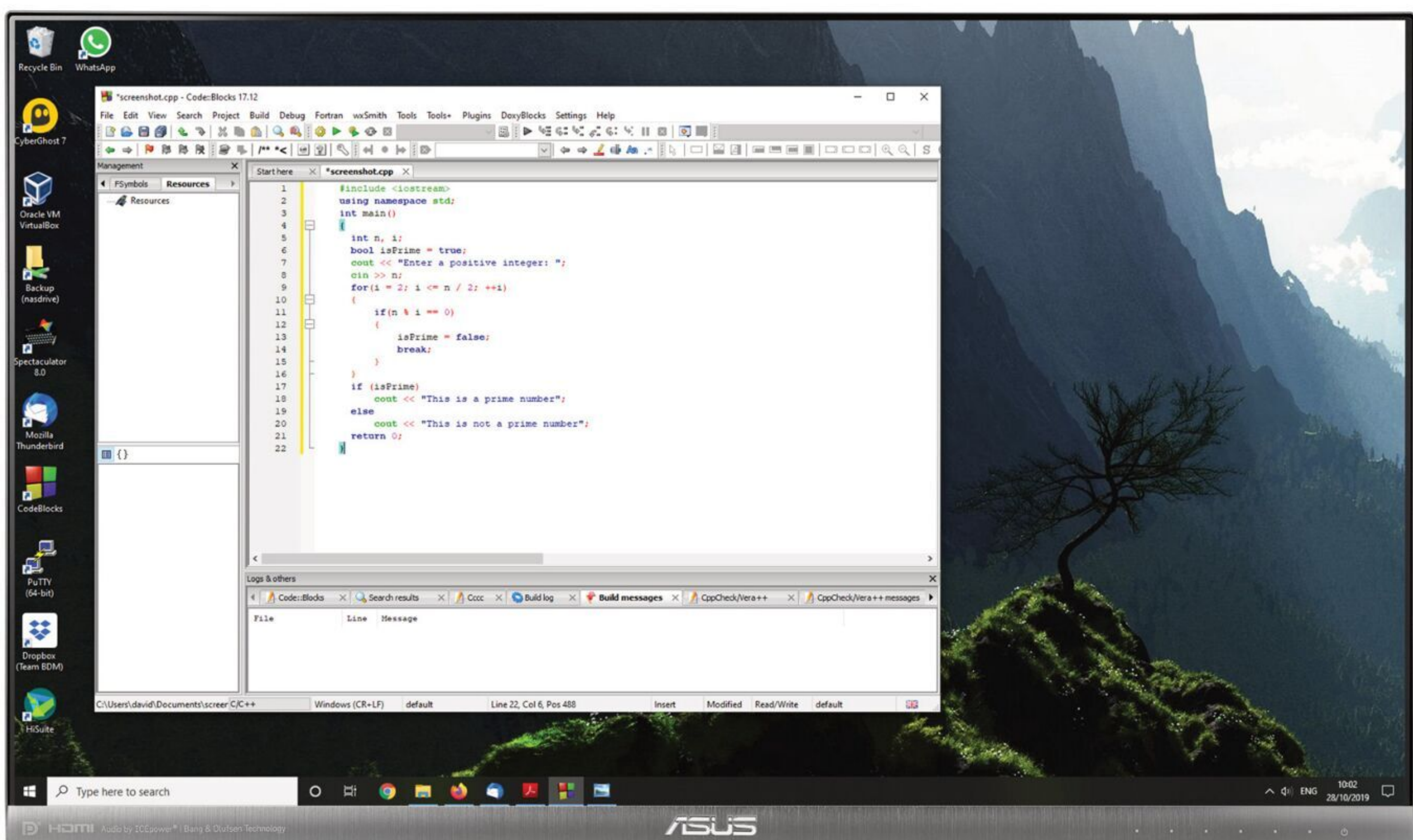
C++'s impact in the digital world is immense. Many of the programs, applications, games, and even operating systems are coded

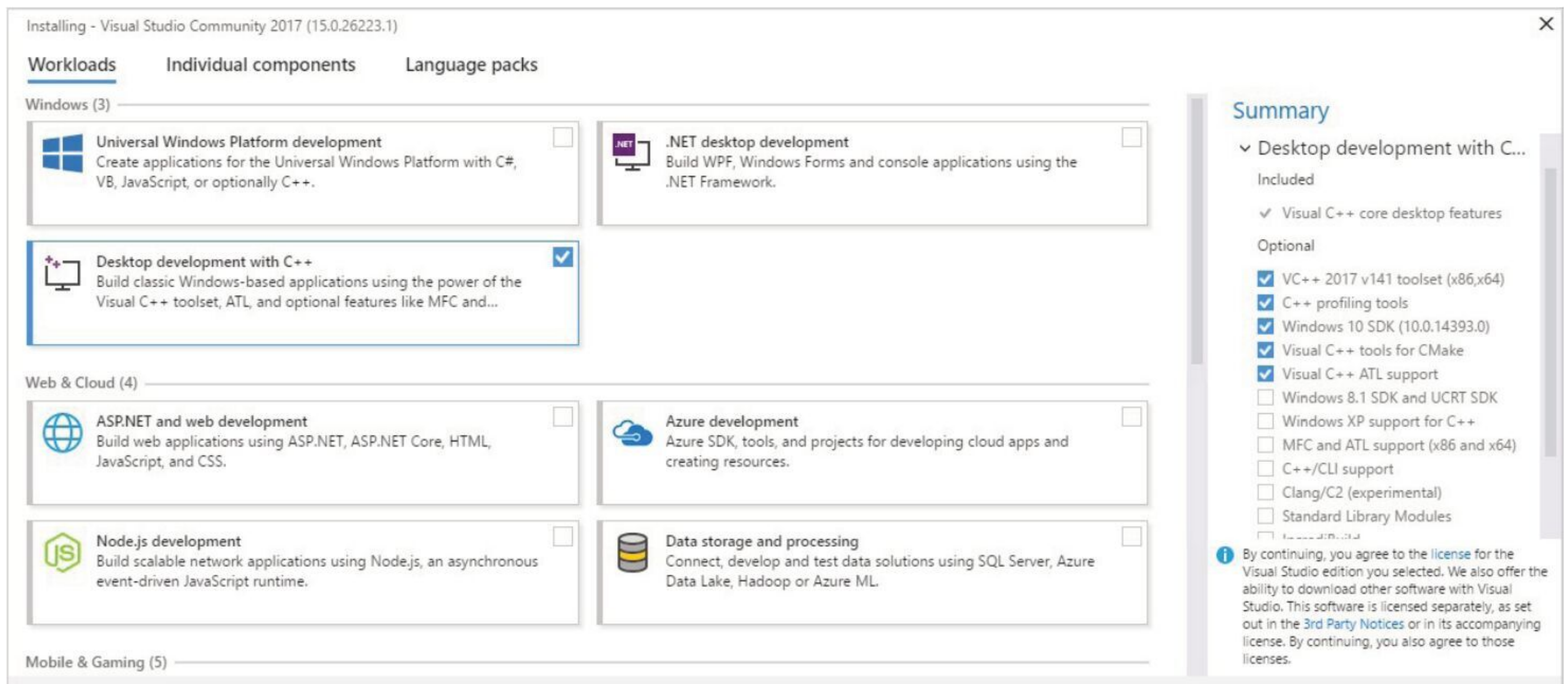
using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you use to surf the Internet is written in C++, as well as Windows 10, Microsoft Office, and the backbone to Google's search engine. Apple's macOS is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

As well as being an easier addition to the core C language, C++ is also extremely efficient and performs well across the board. 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.



C++ code is much faster than other programming languages.





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

C++ puts the developer in a much wider world of coding. By mastering C++, you will find yourself being able to develop 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 re-learn anything specific.

You will discover, as you become a more advanced coder, that many of the developers in various coding jobs around the world tend to use pre-designed development engines. For example, when creating games, the likes of Bethesda, the team behind Oblivion and Skyrim, utilise a 3D game engine called The Creation Engine. This enables the team to quickly create animations, characters, items, terrains, rooms, and just about everything else you'd see in the game. The engine itself has been modified to make the most of the current graphics card hardware, and computer or console processing power. These engines are mostly written in C++, and when making

improvements to the engine, or when creating a new game, if the developers want to add something that the engine can't do, they will use C++ to create the new content or link between two different engines. The end result, of course, is a game that contains the latest graphical technology, while being seamlessly bound together with some pretty clever C++ coding.

Getting to use C++ is quite easy, 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.

So, to answer the question of Why C++, 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 have mastered.



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

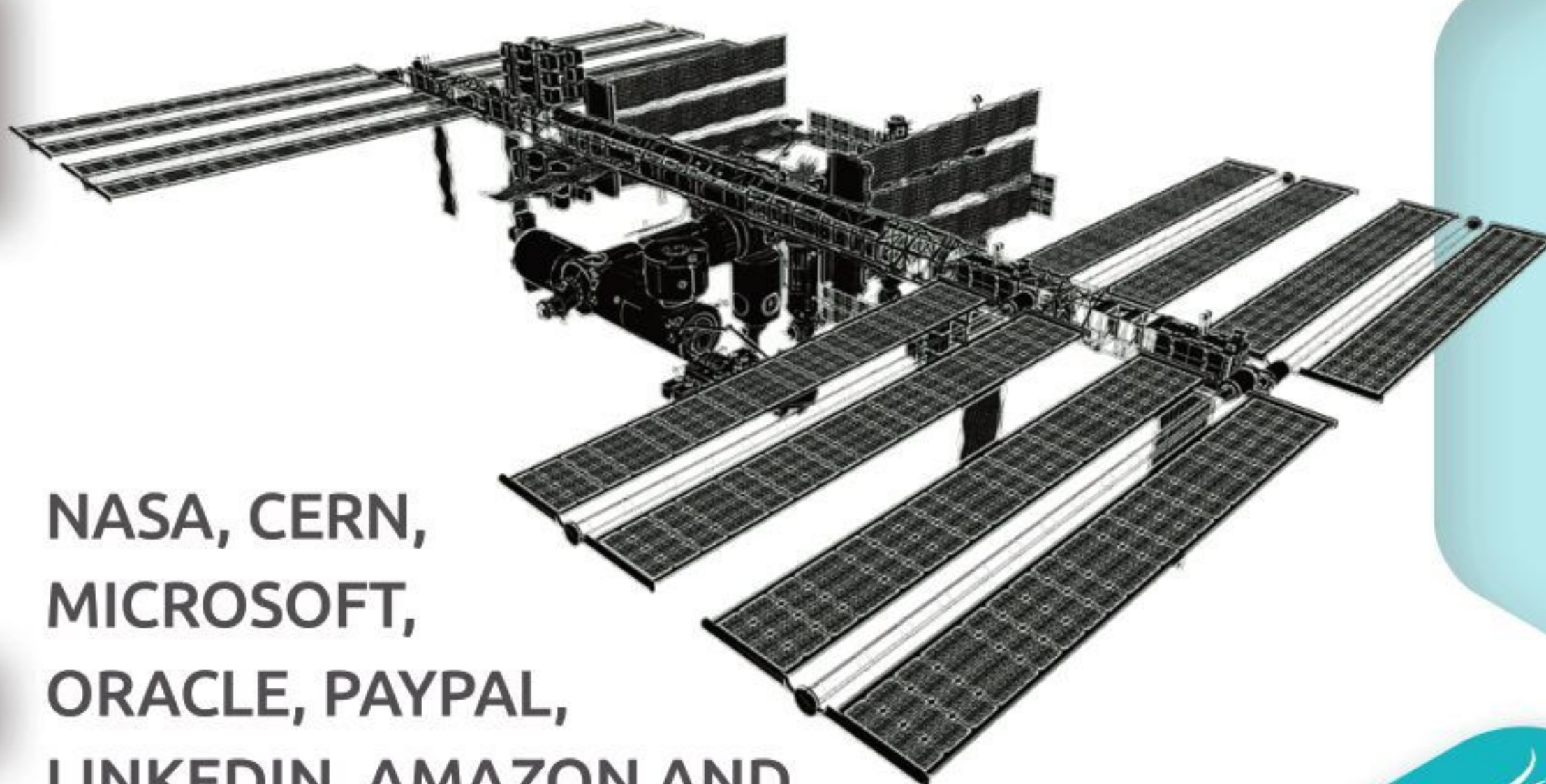




C++ is one of the top programming languages in the industry. It's quick, powerful, and used by nearly every major tech and gaming company in the world. Here's some interesting facts about the rather wonderful C++.



C++ is one of the predominant programming languages for the development of all kinds of technical and commercial software.

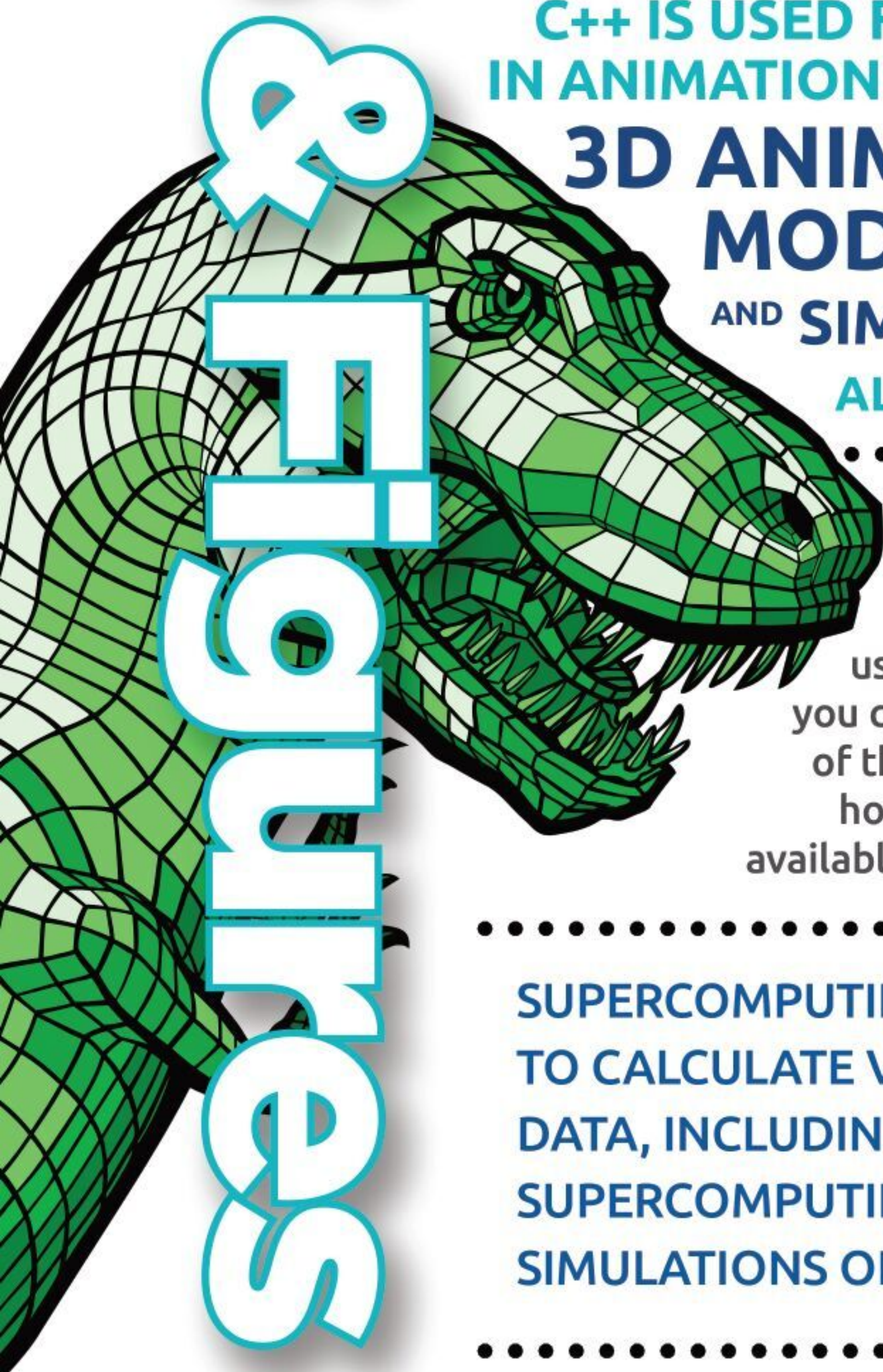


NASA, CERN, MICROSOFT, ORACLE, PAYPAL, LINKEDIN, AMAZON AND THE MILITARY USE C++.

C++ IS USED FREQUENTLY IN ANIMATION PROCESSES.

3D ANIMATION, MODELLING, AND SIMULATIONS

ALL UTILISE C++.



A lot of the Linux operating system is coded using C++, therefore you could say that most of the world's Internet hosting servers are available thanks to C++.



SUPERCOMPUTING USES C++ CODE TO CALCULATE VAST AMOUNTS OF DATA, INCLUDING THE NASA-BASED SUPERCOMPUTING FARM THAT RUNS SIMULATIONS OF THE UNIVERSE.



More than 70% of all trading is known as High Frequency Trading (HFT), and the software responsible is written in C++ to make use of its high speed.

Sources: ITJobsWatch,
Google, Quora, Stack
Exchange, eduCBA,
StackOverflow,
stackShare,
AppDynamics

C++ Facts & Figures



The Xbox
operating system
uses C++ as its
backbone.



Many database applications
are built using C++, such
as MySQL, it's also used
by Wikipedia, Yahoo and
YouTube.



Gaming is one of the
biggest users of C++.
It handles the
complexities of 3D games,
supports multiplayer
options, and enables
intensive CPU and
GPU hardware
functions.



Google Chrome, Mozilla's
Firefox, and even Microsoft's
Edge web browsers are
coded in C++.



C++ is
used in computer
networking as the
main code behind the
Programmable Logic
Controller, connecting
servers, processors, other
hardware, and
even robotics.

THE
90th
PERCENTILE SALARY
FOR A UK-BASED C++
DEVELOPER IS
£120,000



Most of Adobe's
stable of products
are developed
using C++.



Windows 95, 98, 2000,
XP, 7, 8.1 and 10, as well as
Microsoft Office, use C++
as the backbone programming
language of choice.



MRI scanning
machines and
Computer Aided Design
all use C++ to help with
the enhanced imagery
produced by these
systems.



The Unreal 4 Engine,
which is coded in C++,
is used for hundreds
of games, including
Fortnite.

.....
**RADAR
PROCESSING
TECHNOLOGIES
USE C++, AS
WELL AS
ADVANCED
MISSILE
DETECTION
SYSTEMS.**
.....

.....
**FLIGHT CONTROL SYSTEMS IN
MODERN AIRCRAFT, INCLUDING
MILITARY AIRCRAFT, USE 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

As most, if not all, operating systems have C++ at their code, it stands to reason that you can learn to program in C++ no matter what OS you're currently using.



☐ 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

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 enhancements for Windows 10.

☐ COMPILER

A compiler is a program that converts 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

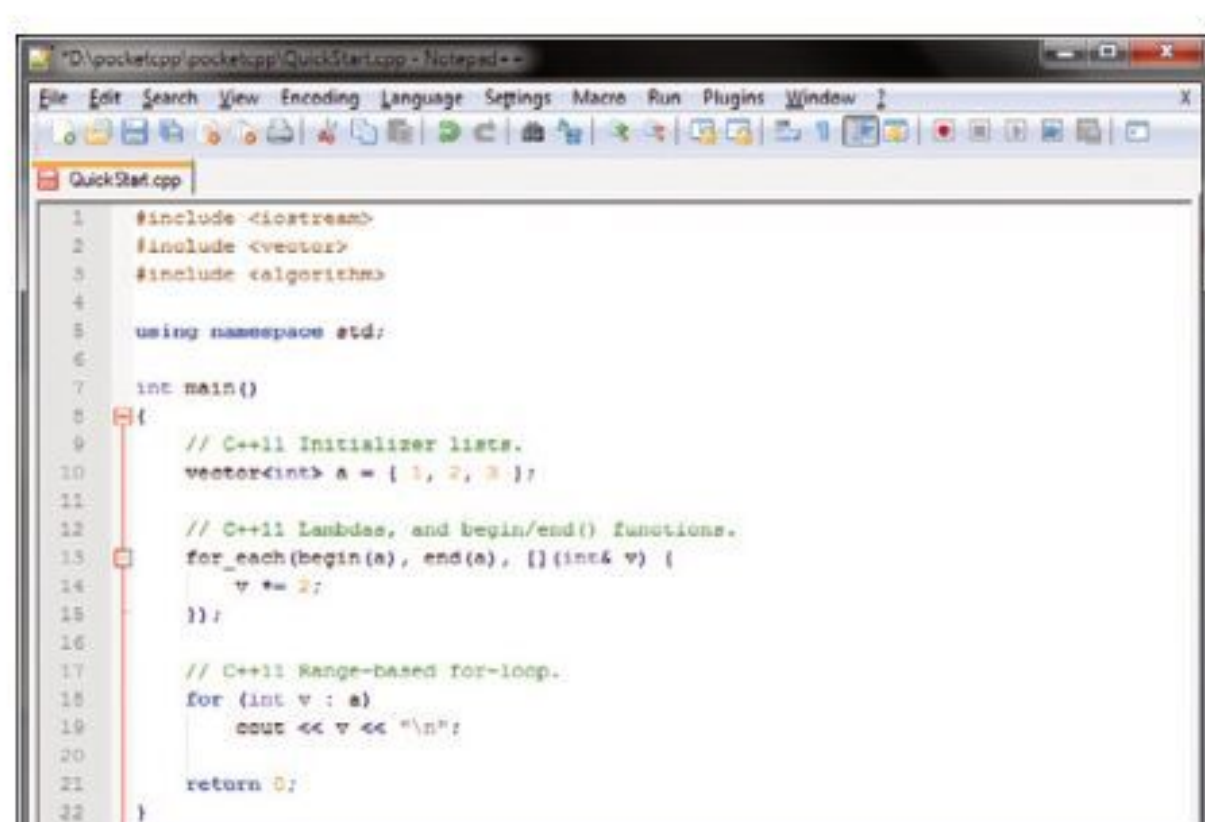
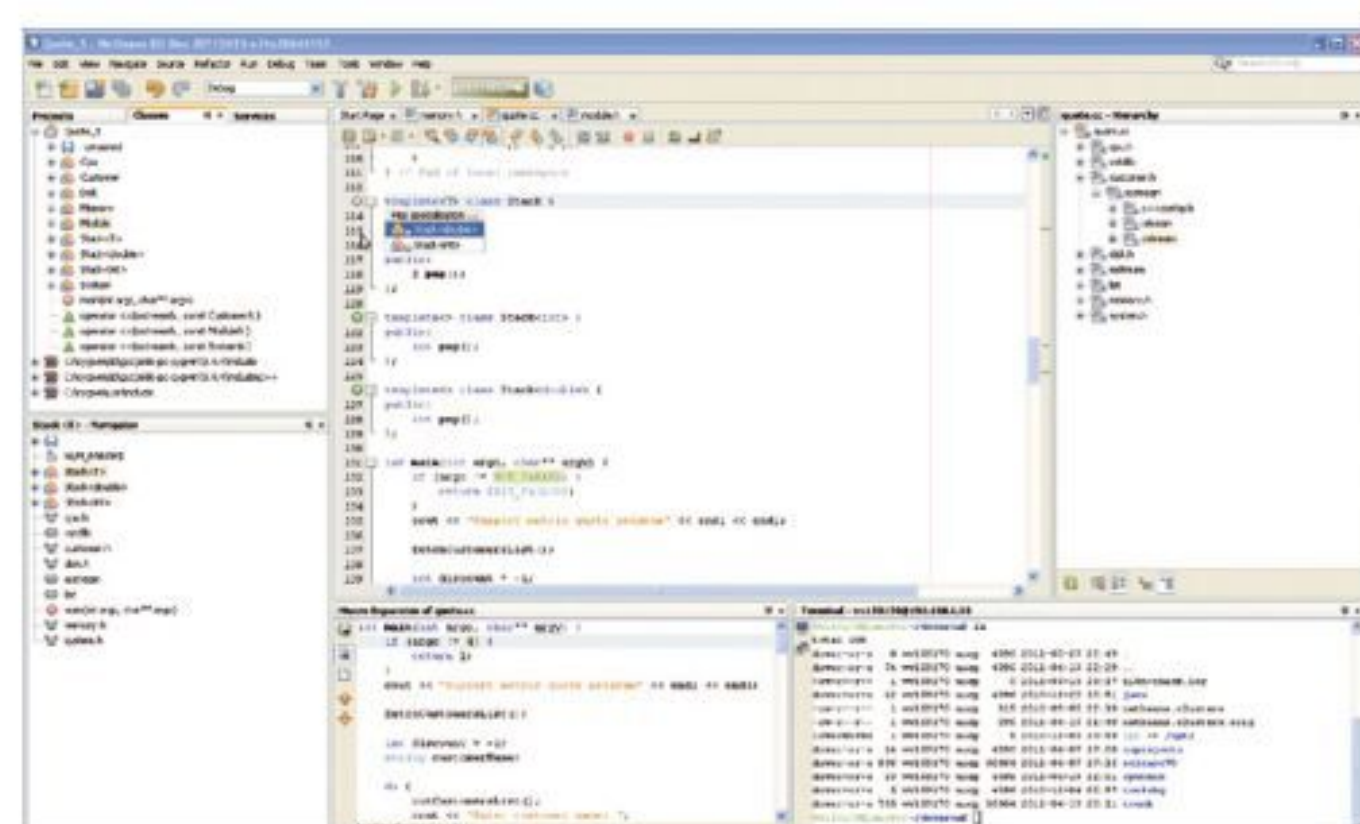
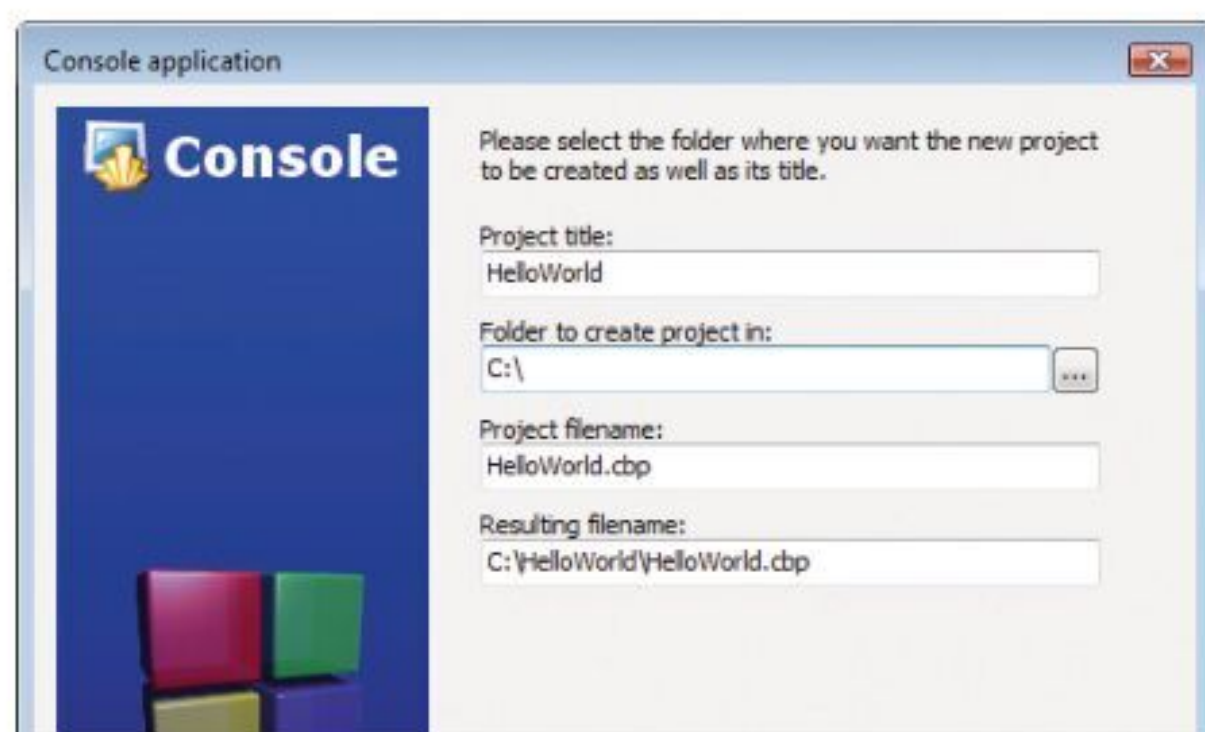
Some programmers much prefer to use a text editor to assemble their C++ code before running it through a compiler. Essentially, you can use any text editor to write code, just save it with a .cpp extension. However, Notepad++ is one of the best code text editors available.

☐ 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

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 in a week. A good C++ coder has spent many years honing their craft, so be patient, start small and keep learning.



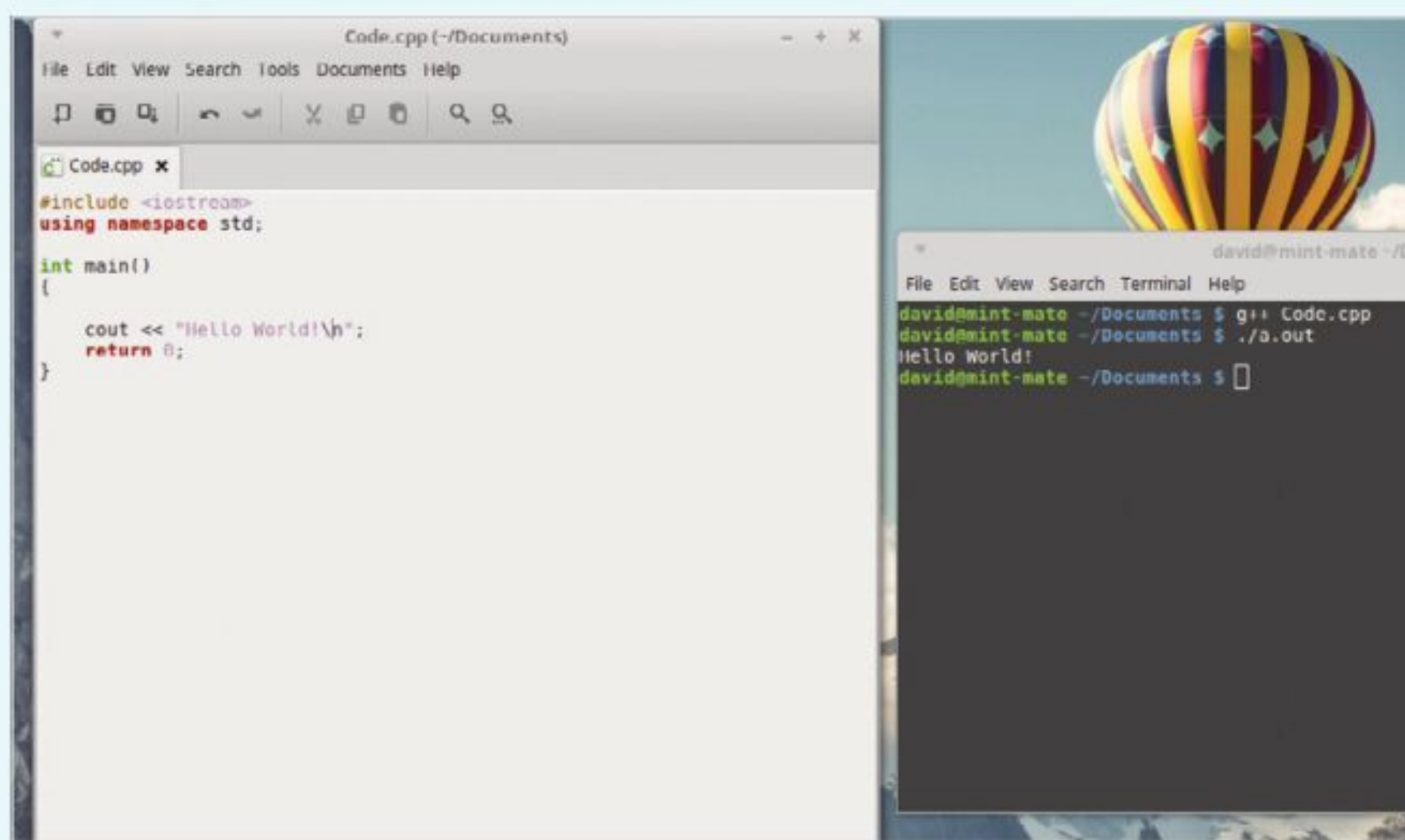


OS SPECIFIC NEEDS

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++.

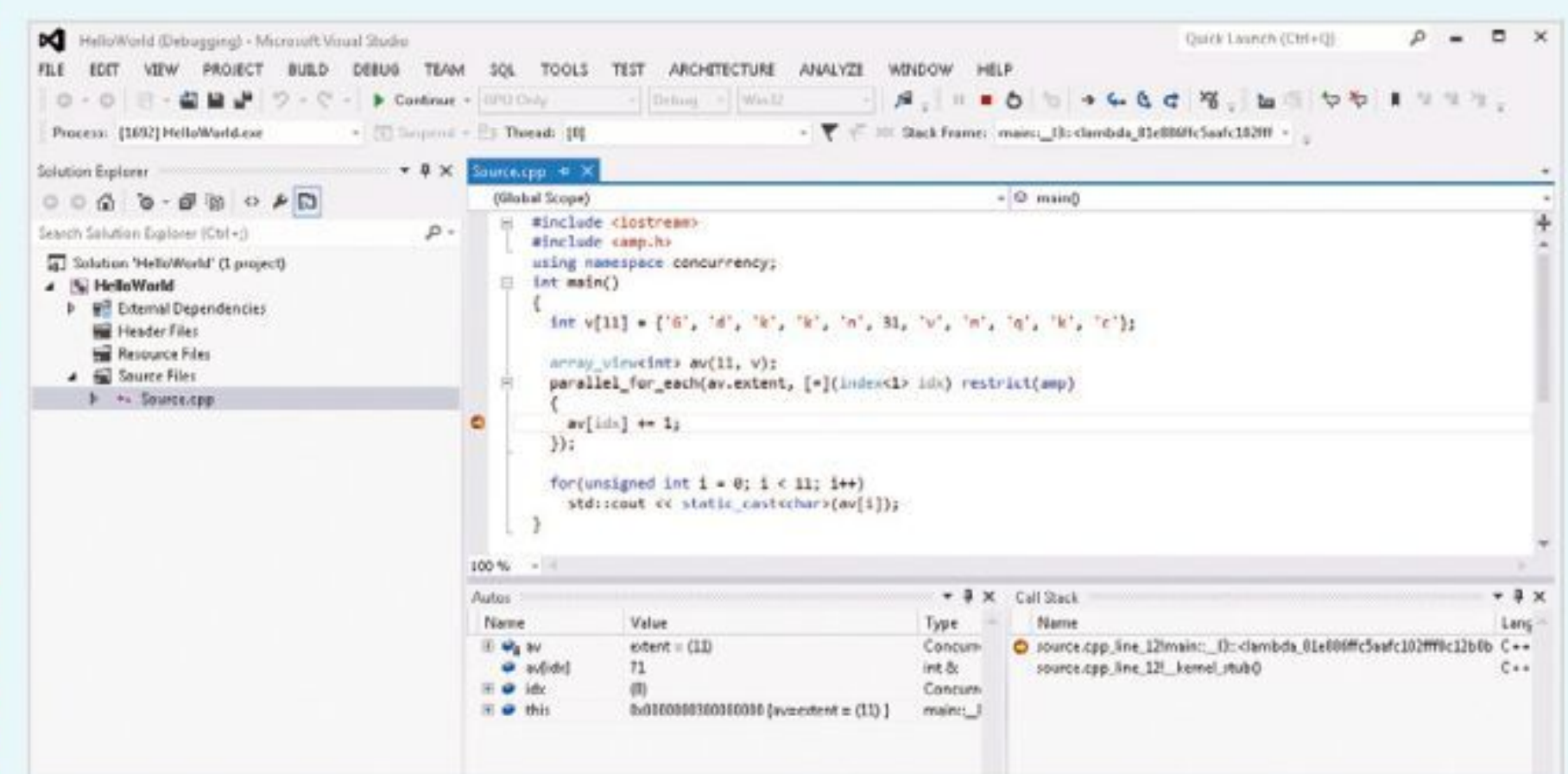
LINUX

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 to type out your C++ code, when it's saved with a .cpp extension, use g++ to compile it.



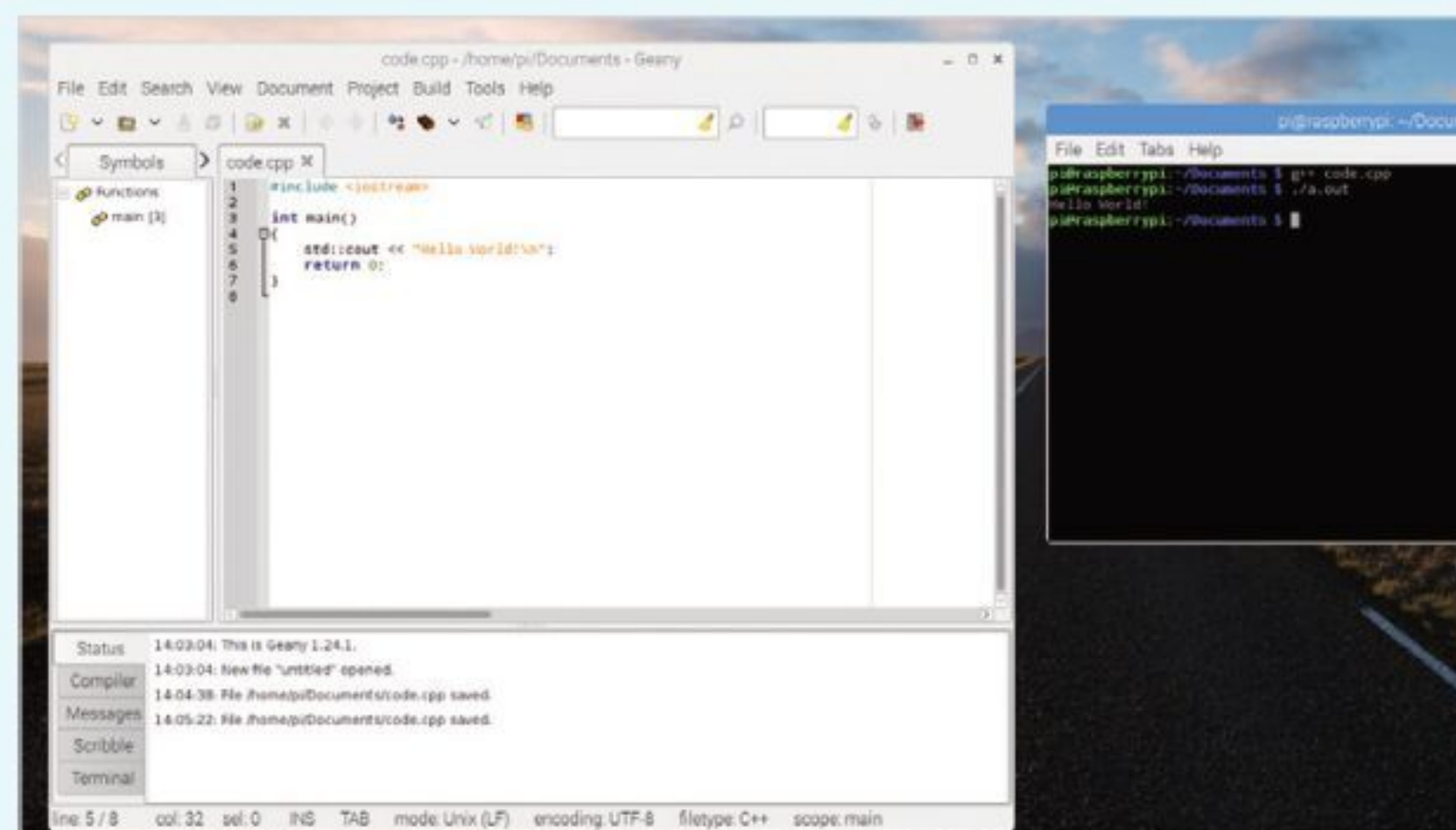
WINDOWS

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



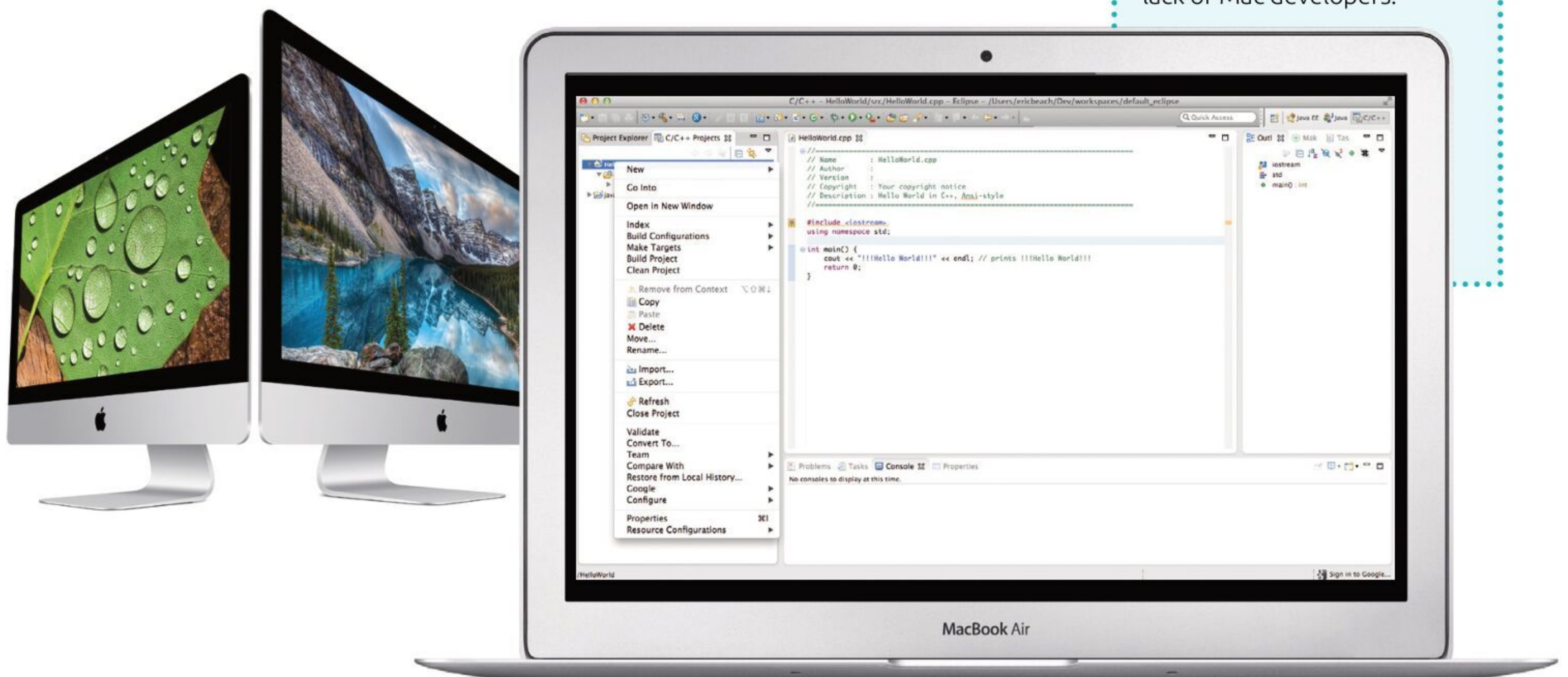
RASPBERRY PI

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 and then compile it with g++, as you would in any other Linux distro.



MAC

Mac owners will need to download and install Xcode, in order 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.





Structure of C++ Code

C++ is an amazing programming language 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!>

As you learn the basics of programming, you will begin to understand the structure of a program. The commands may be different, but you will start to see how the code works.

C++

Danish student Bjarne Stroustrup invented C++ in 1979, as a part of his Ph.D. 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 can look complex, but once you get familiar with it you'll begin to see how it flows. 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
# C++ *newcode.cpp x
#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 lies within the main body of the code.

```
*newcode.cpp (~ /D
File Edit View Search Tools Documents Help
# C++ *newcode.cpp x
#include <iostream>
int main()
```

BRACES

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

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

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

STD

In C++, STD 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 x
#include <iostream>
using namespace std;

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

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 x
#include <iostream>

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

<<

The two chevrons used here are insertion operators. This means that, whatever follows, the chevrons are to be inserted into the `std::cout` statement. In this case, they are the words '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 x
#include <iostream>

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

OUTPUTS

Leading on, ("Hello World!") is the part that 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");
```

; AND }

Finally you will notice 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.

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

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




How to Set Up C++ in Windows

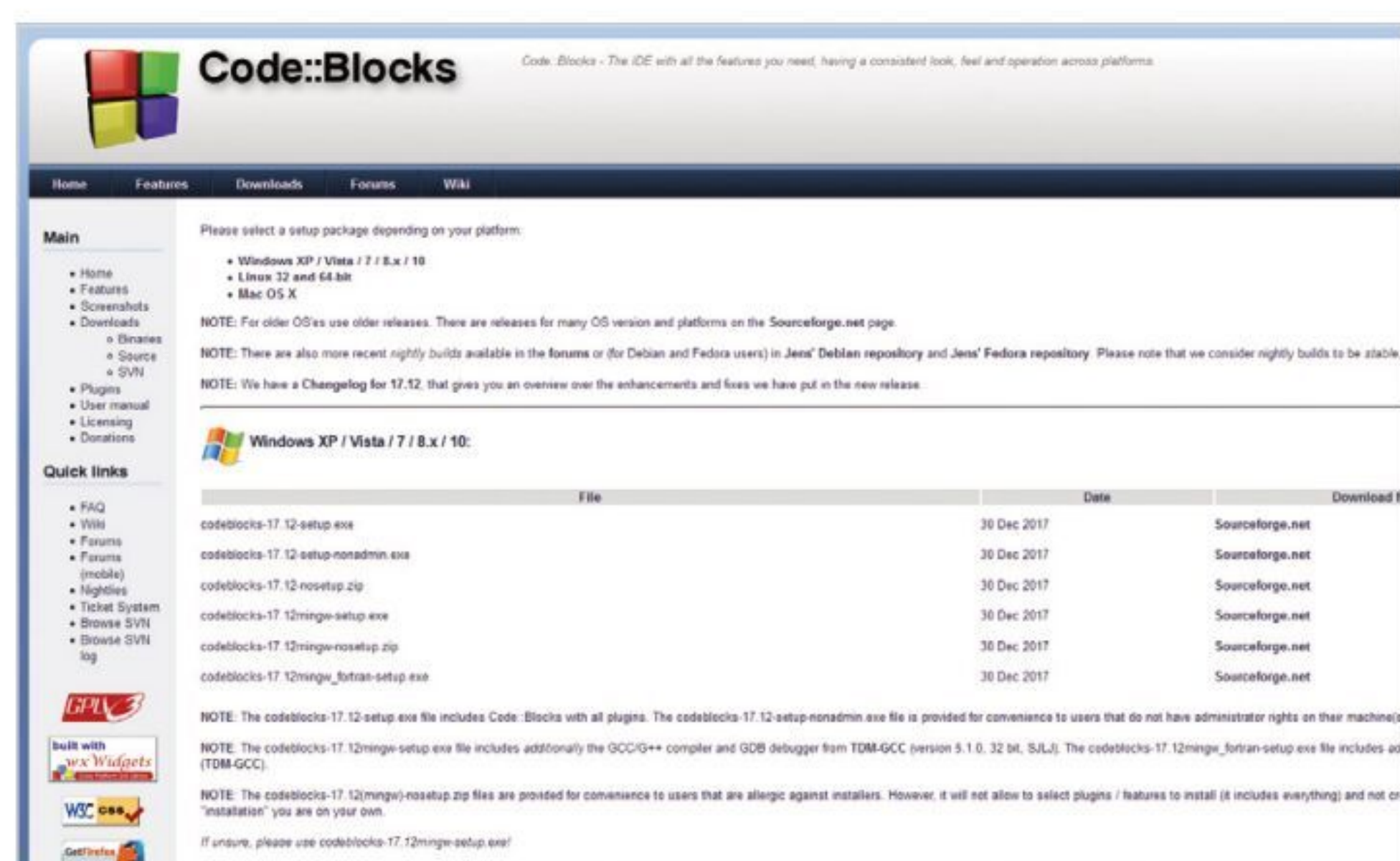
Windows users have a wealth of choice when it comes to programming in C++. There are loads of IDEs and compilers available, including Visual Studio from Microsoft. However, in our opinion, the best C++ IDE to begin with is Code::Blocks.

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.



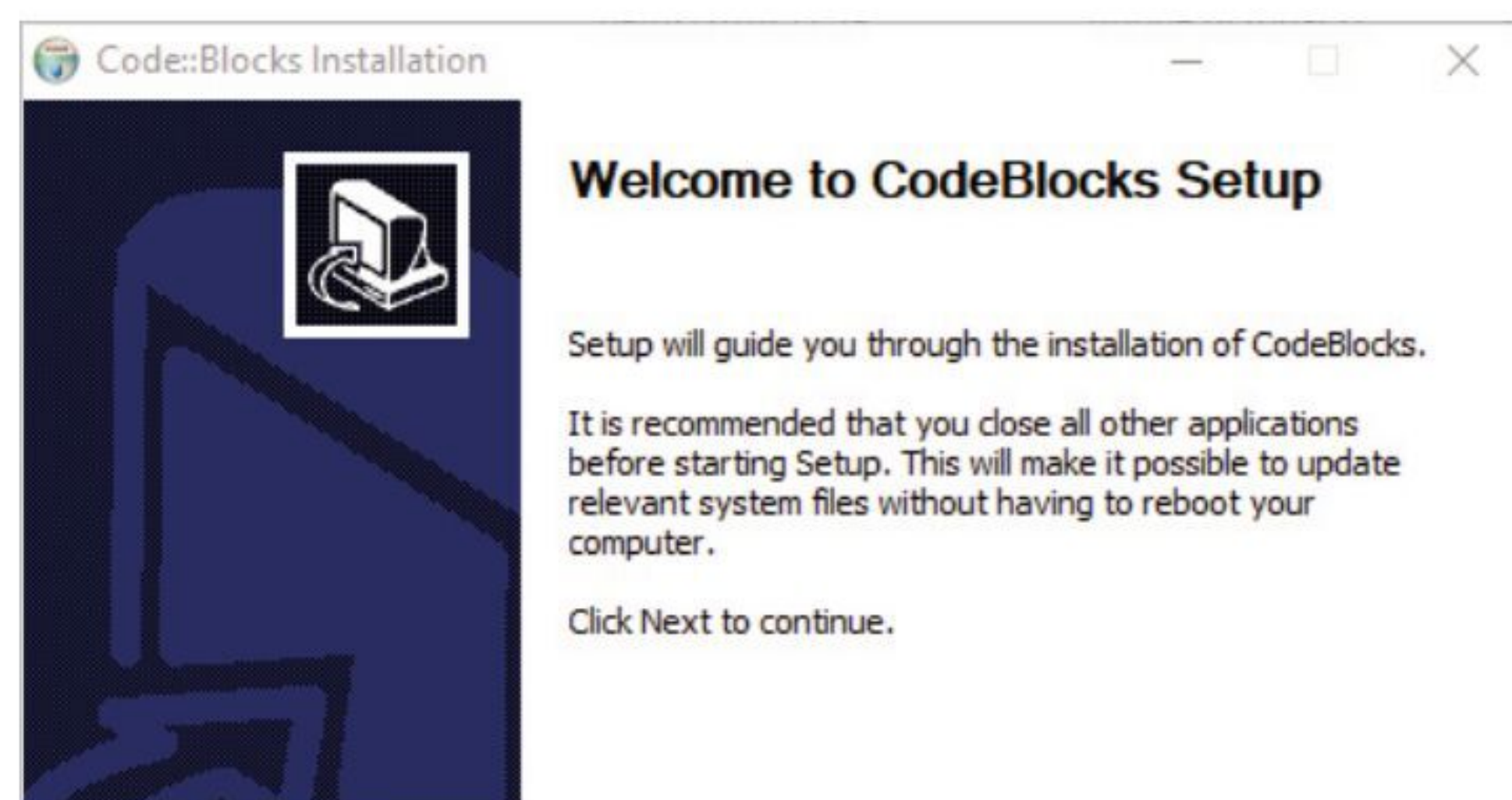
STEP 2

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).



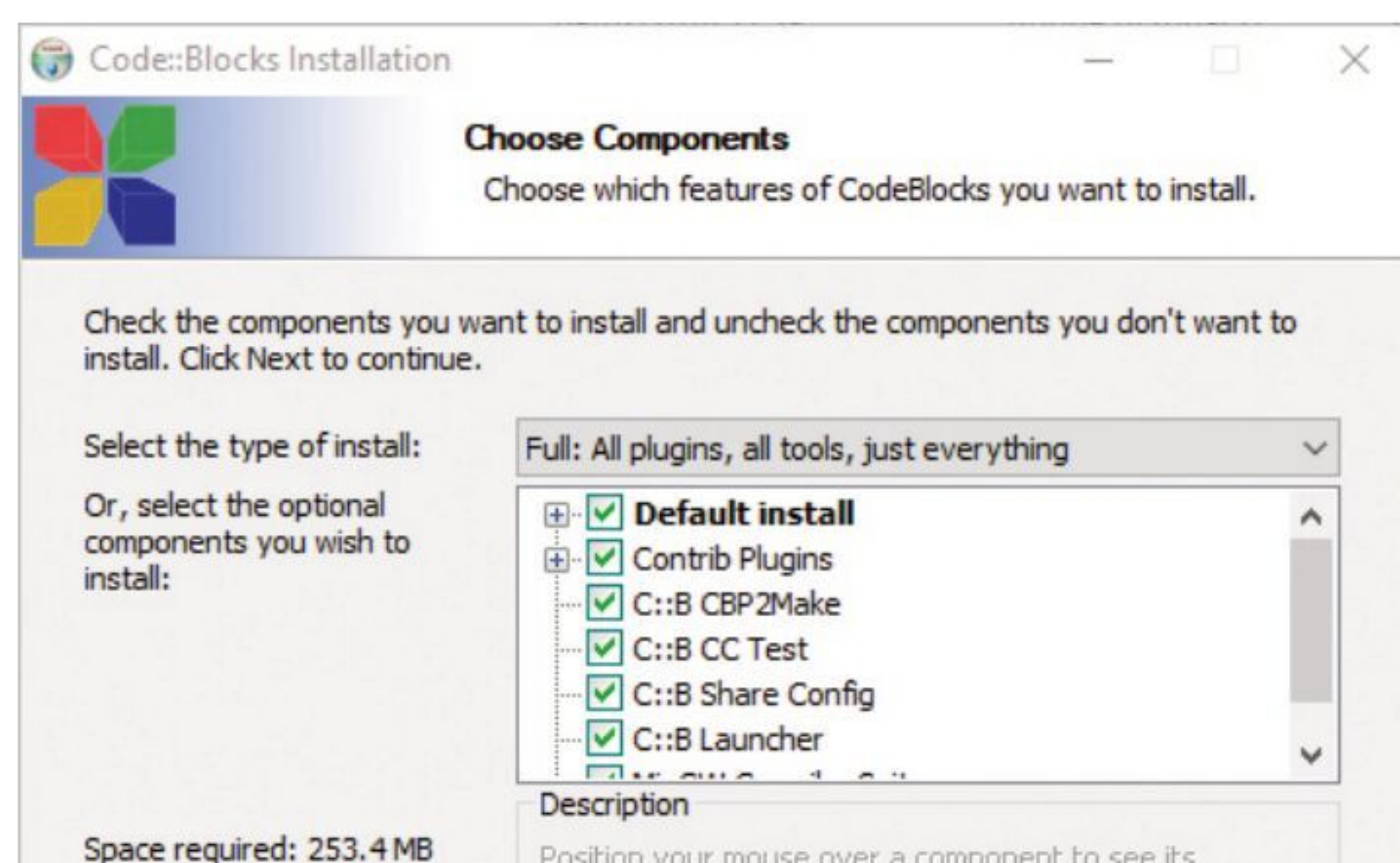
STEP 3

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.



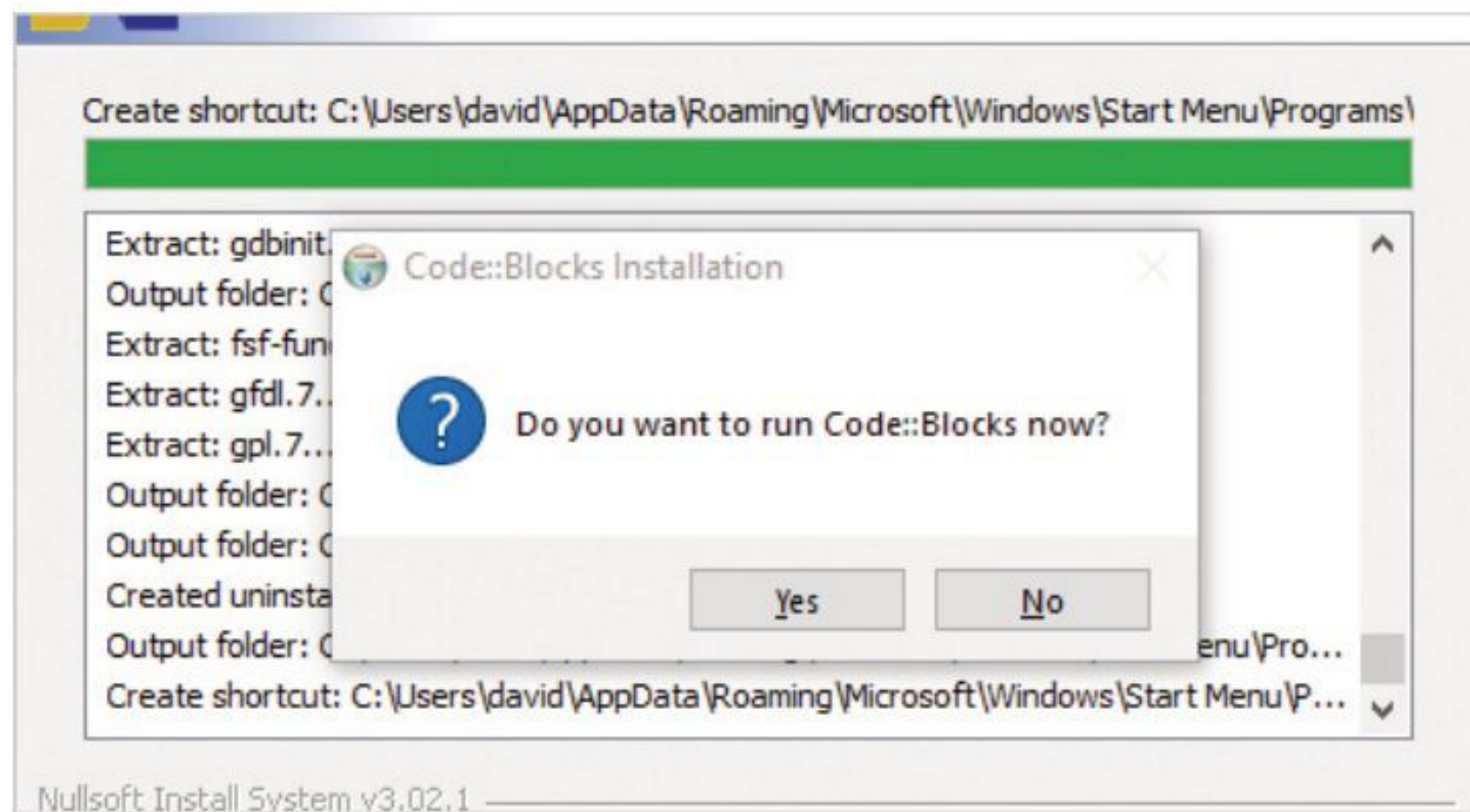
STEP 4

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.

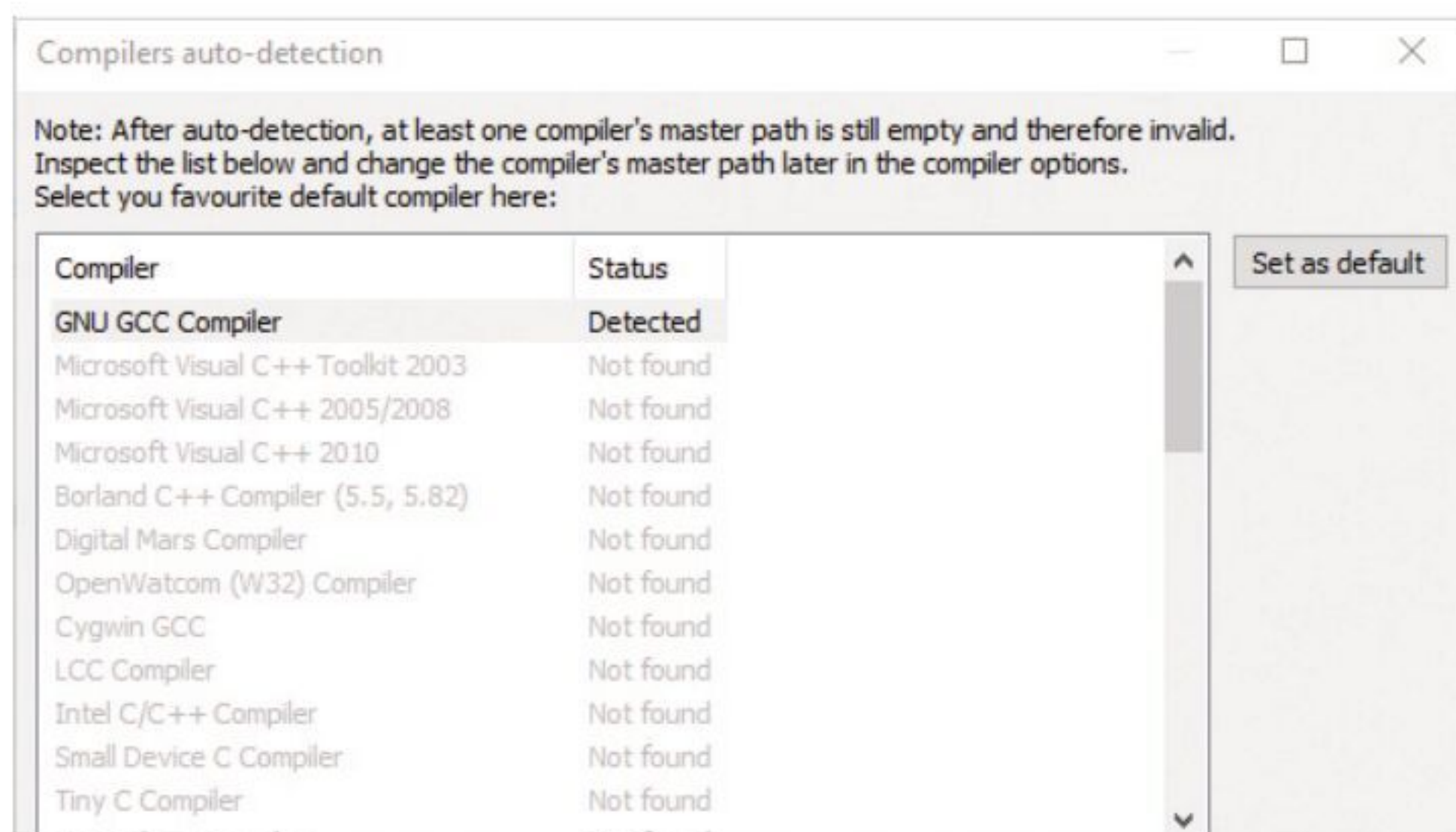


STEP 5

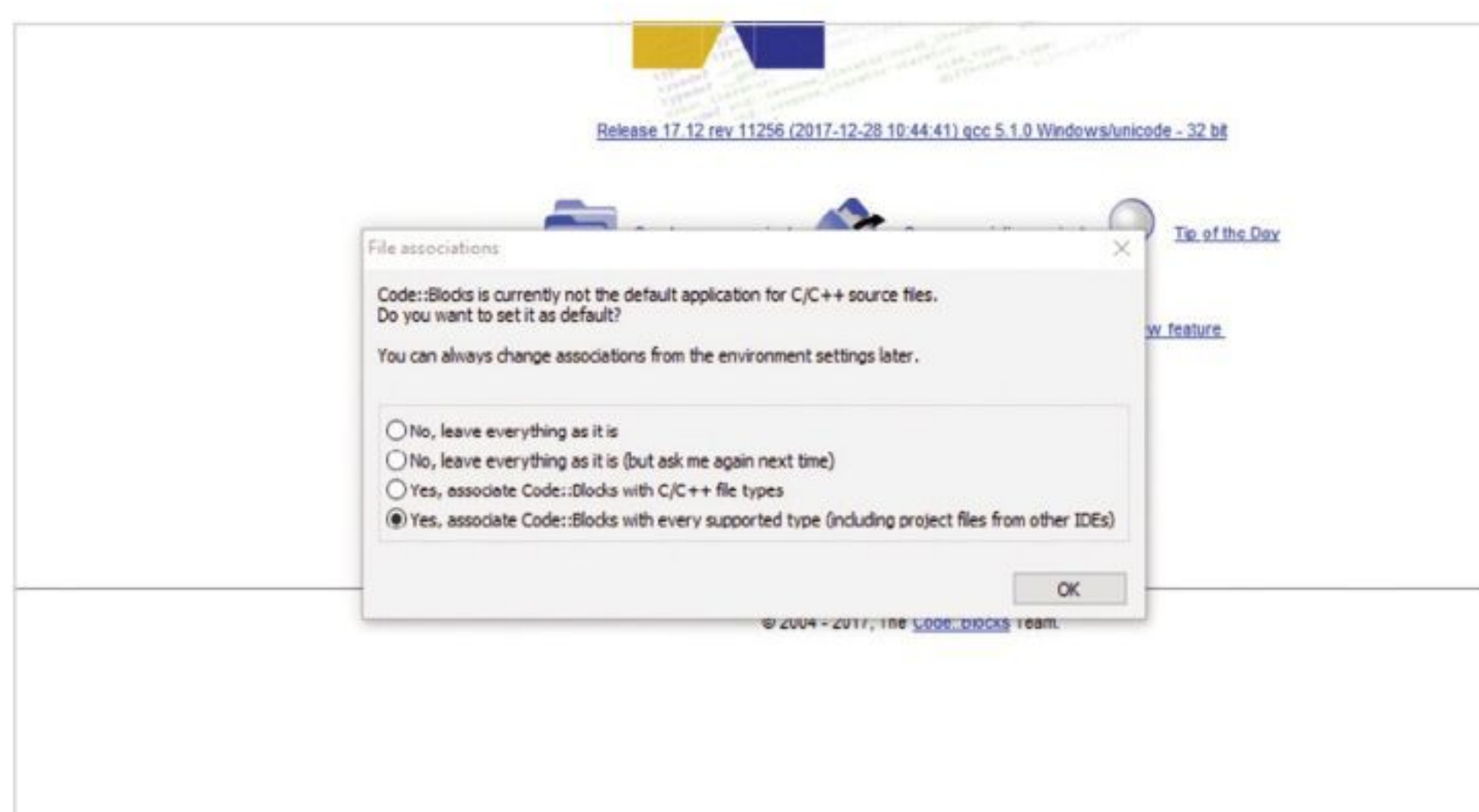
Next choose an install location for the Code::Blocks files. It's your choice but the default is generally sufficient (unless you have any special requirements of course). When you click Next, the install begins; when it's finished a notification pops up asking you if you want start Code::Blocks now,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

**STEP 6**

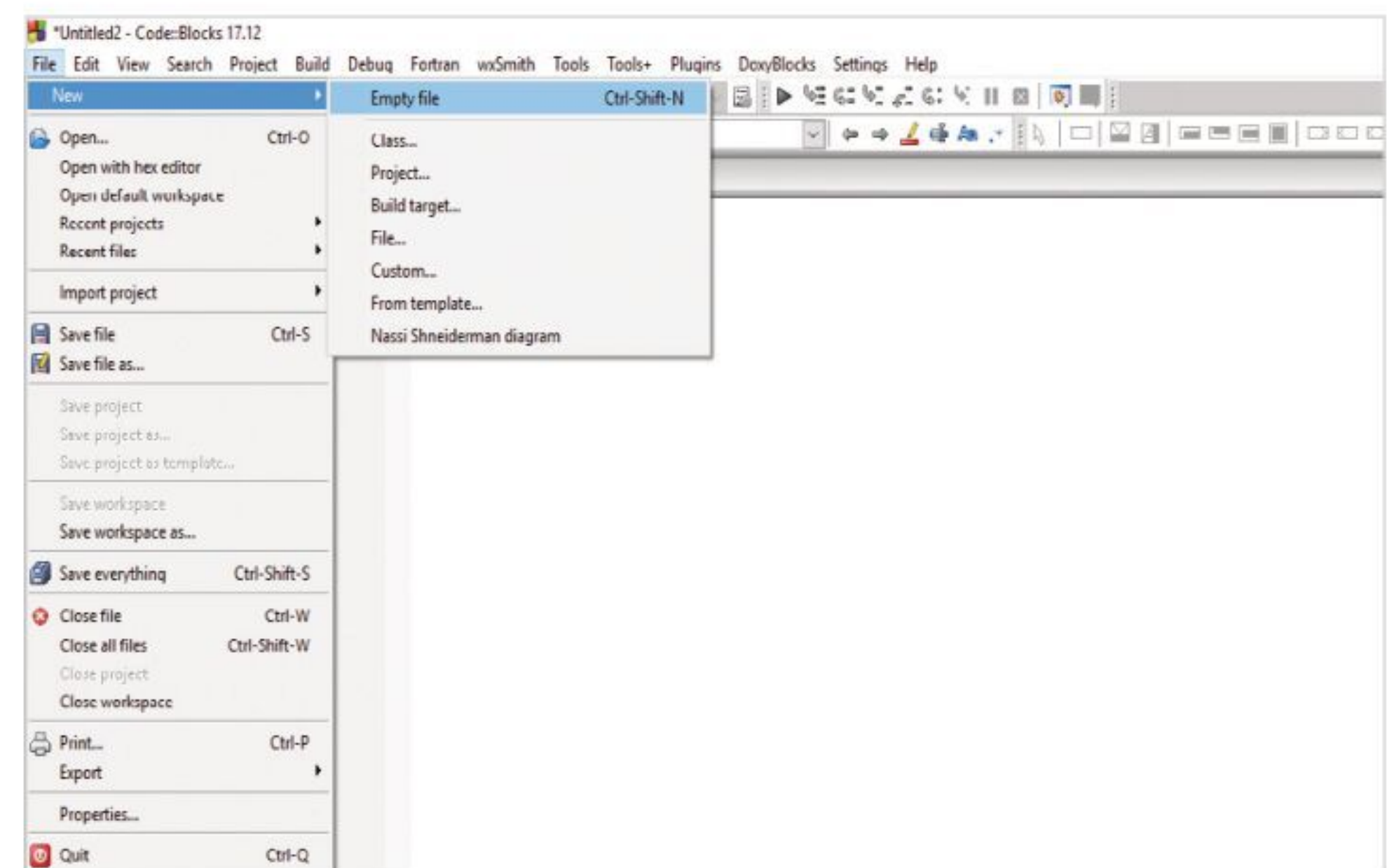
The first time Code::Blocks loads it runs an autodetect for any C++ compilers you may already have installed on your system. If you don't have any, click on the first detected option: GNU GCC Compiler and click the Default button to set it as the system's C++ compiler. Click OK when you're ready to continue.

**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.

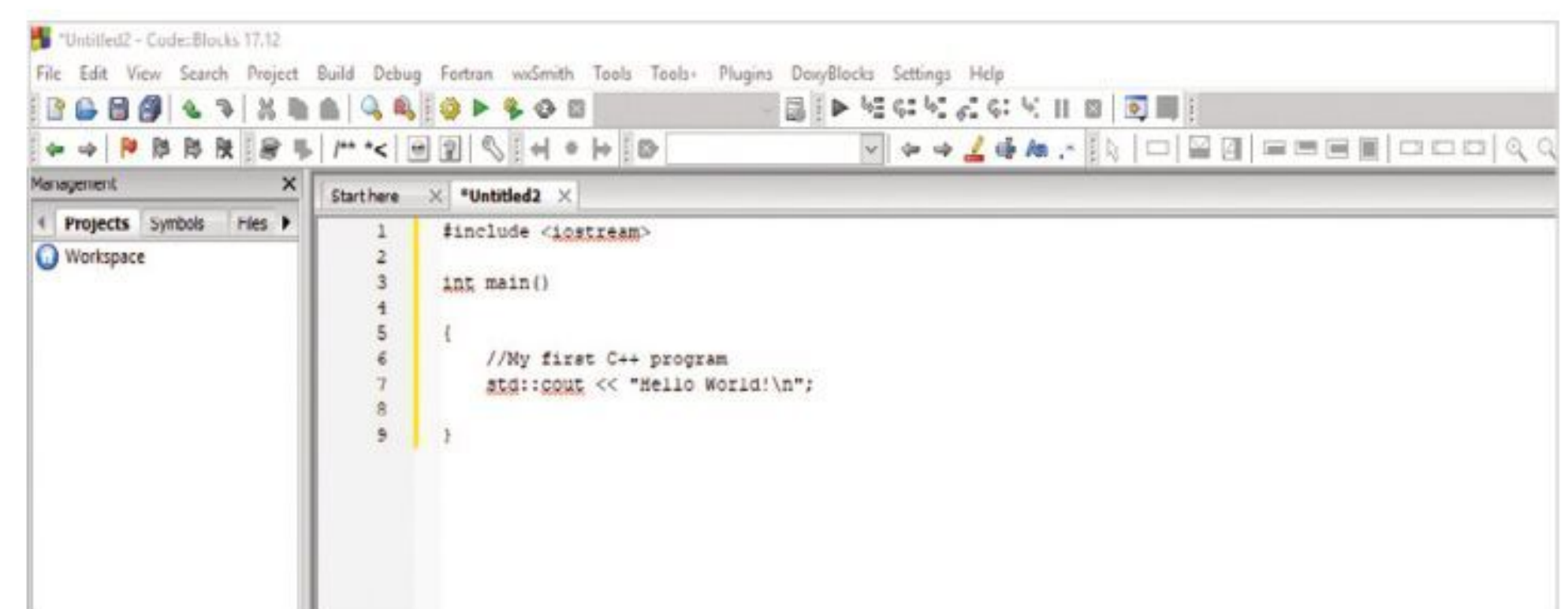
**STEP 9**

In the new window, enter the following:

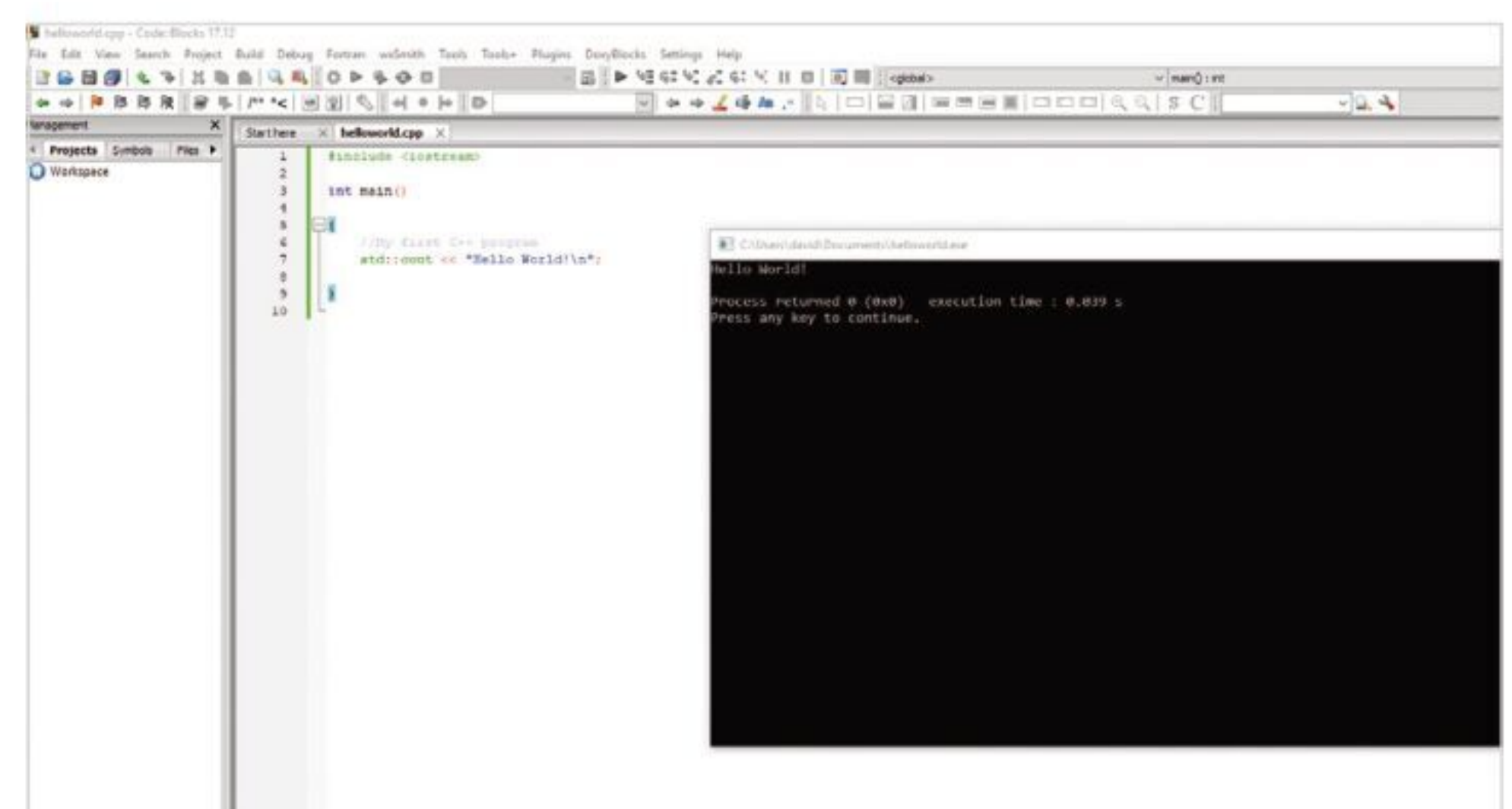
```
#include <iostream>

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

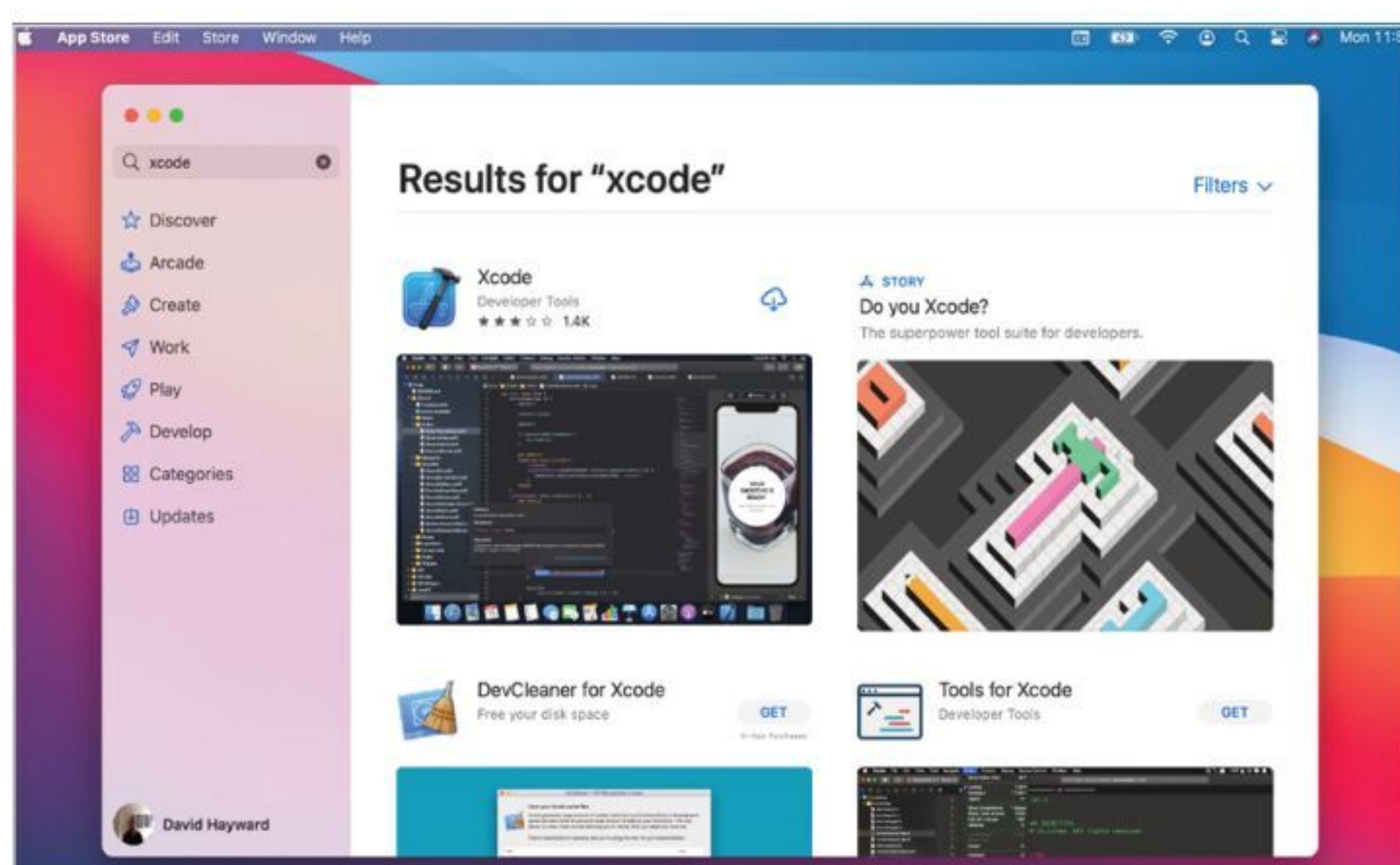
To begin C++ coding on a Mac you can use Apple's own developer platform: Xcode. This is a free, full featured IDE that's designed to create native Apple apps. However, it's also able to be used to create C++ code relatively easily.

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 we can use it for C++ too.

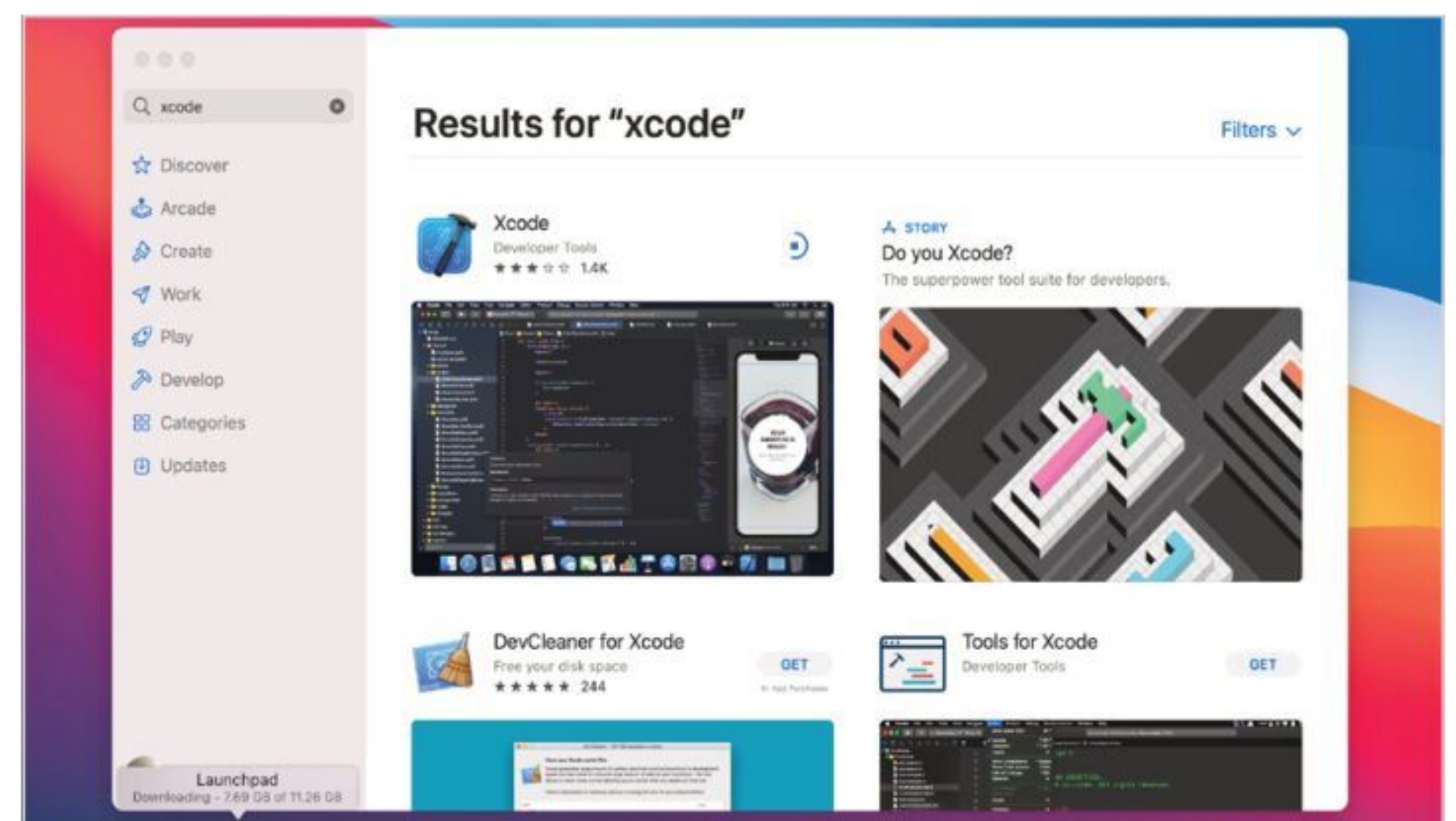
STEP 1

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 first option, Xcode, that you need to click on.



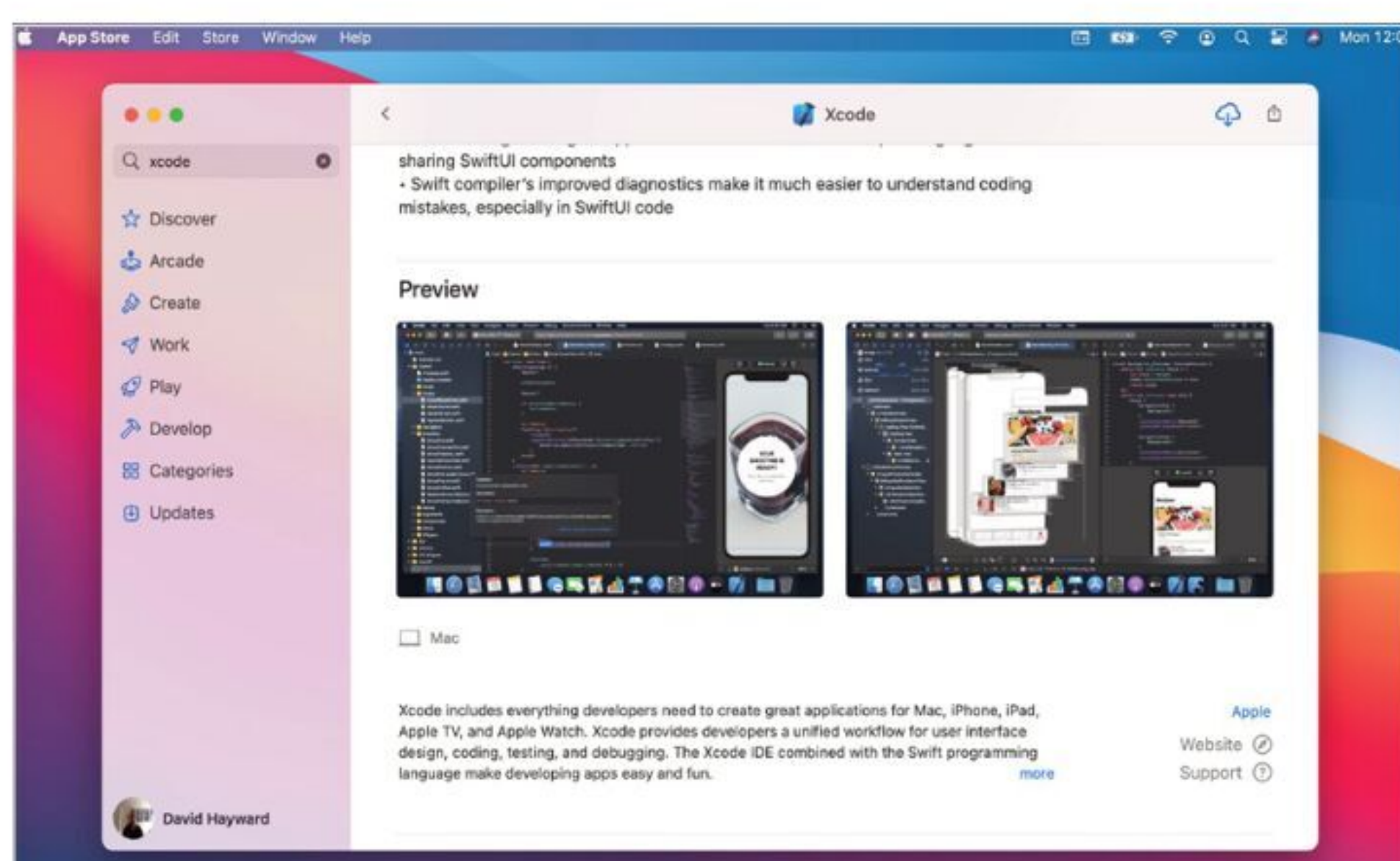
STEP 3

When you're ready, click on the Get or cloud icon button which will install the Xcode app. Enter your Apple ID, and Xcode will begin to download and install. It may take some time depending on the speed of your Internet connection, as Xcode is in excess of 11GB.



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.



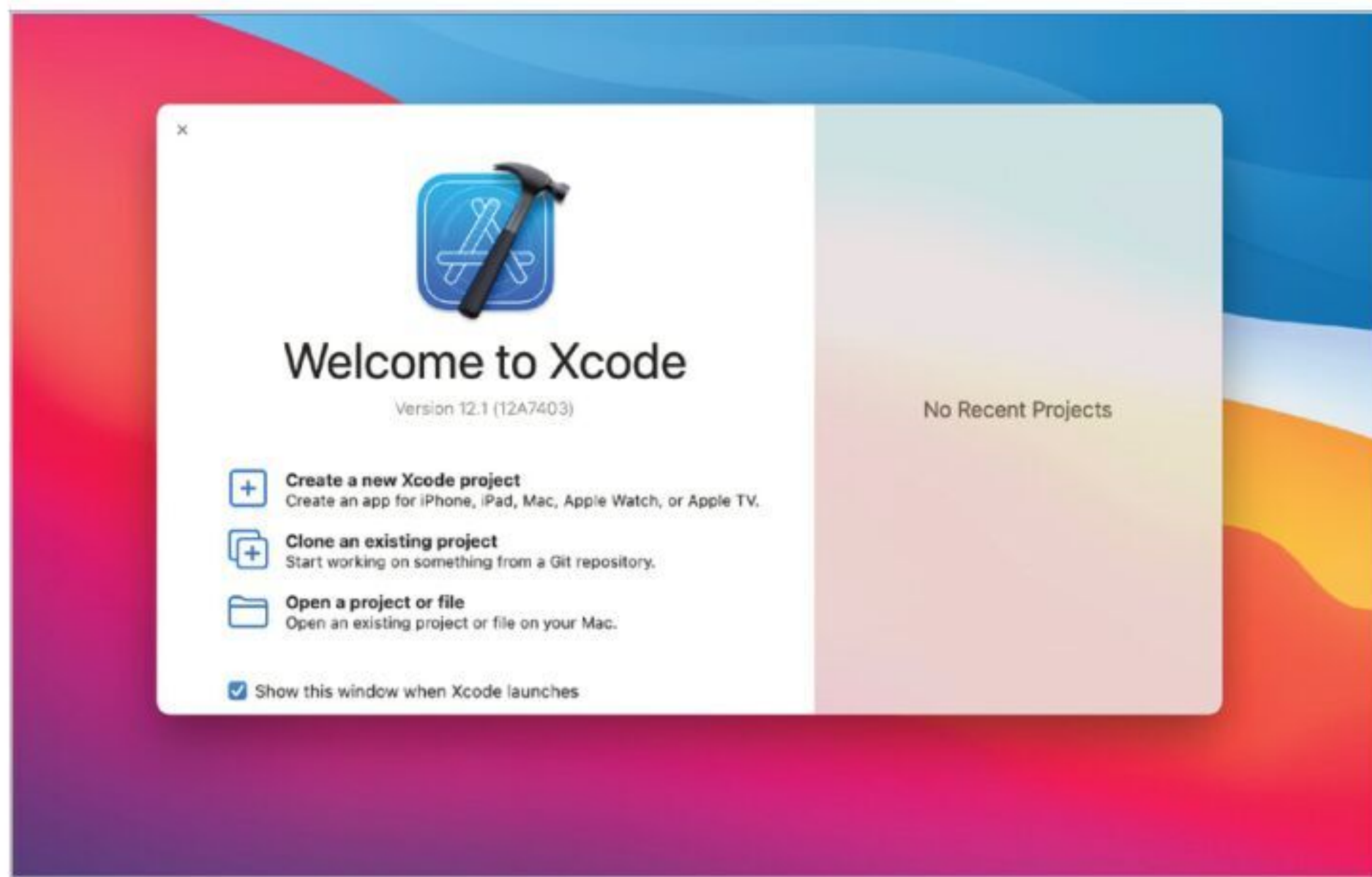
STEP 4

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 will begin to install additional components.

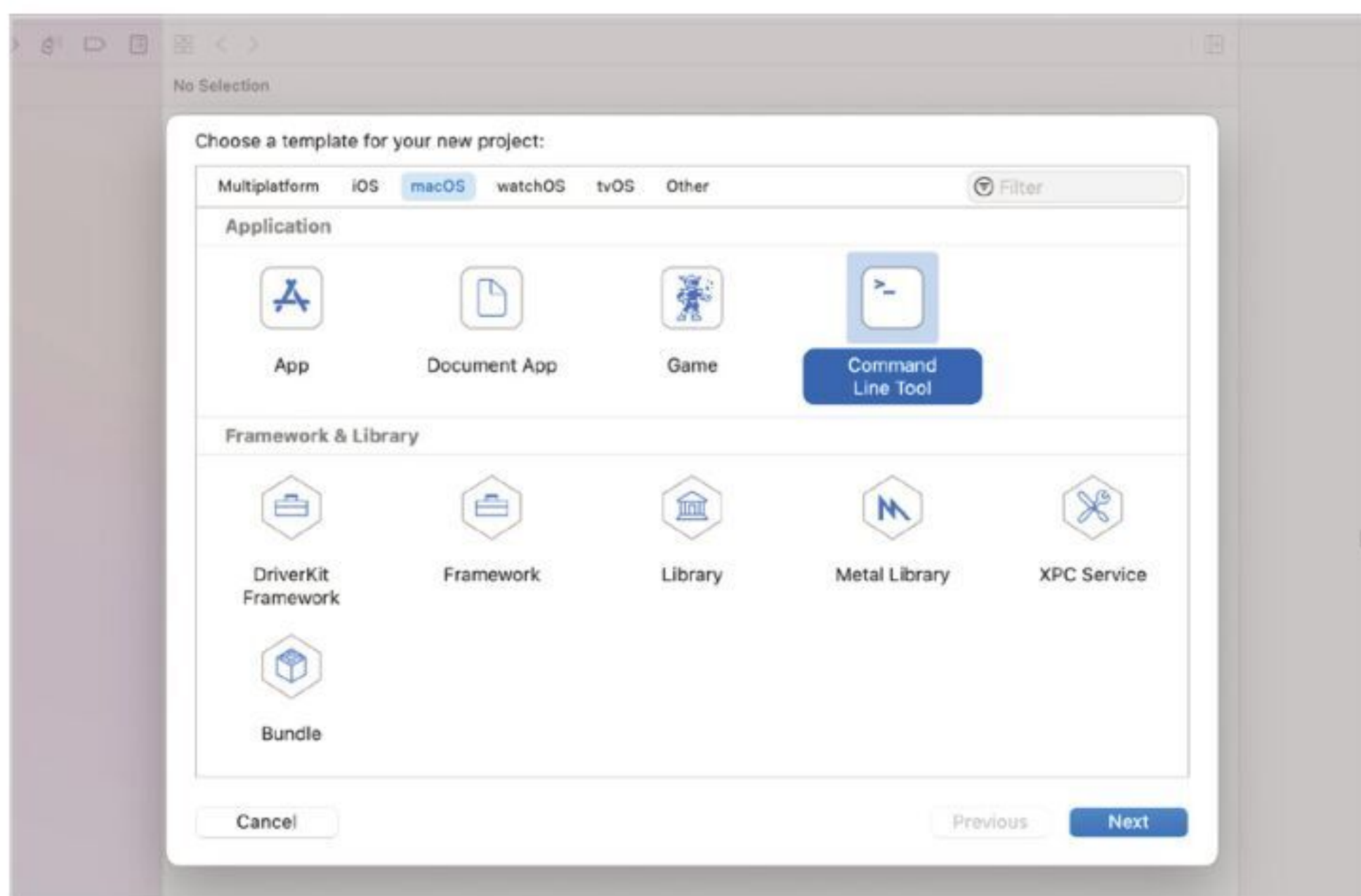


STEP 5

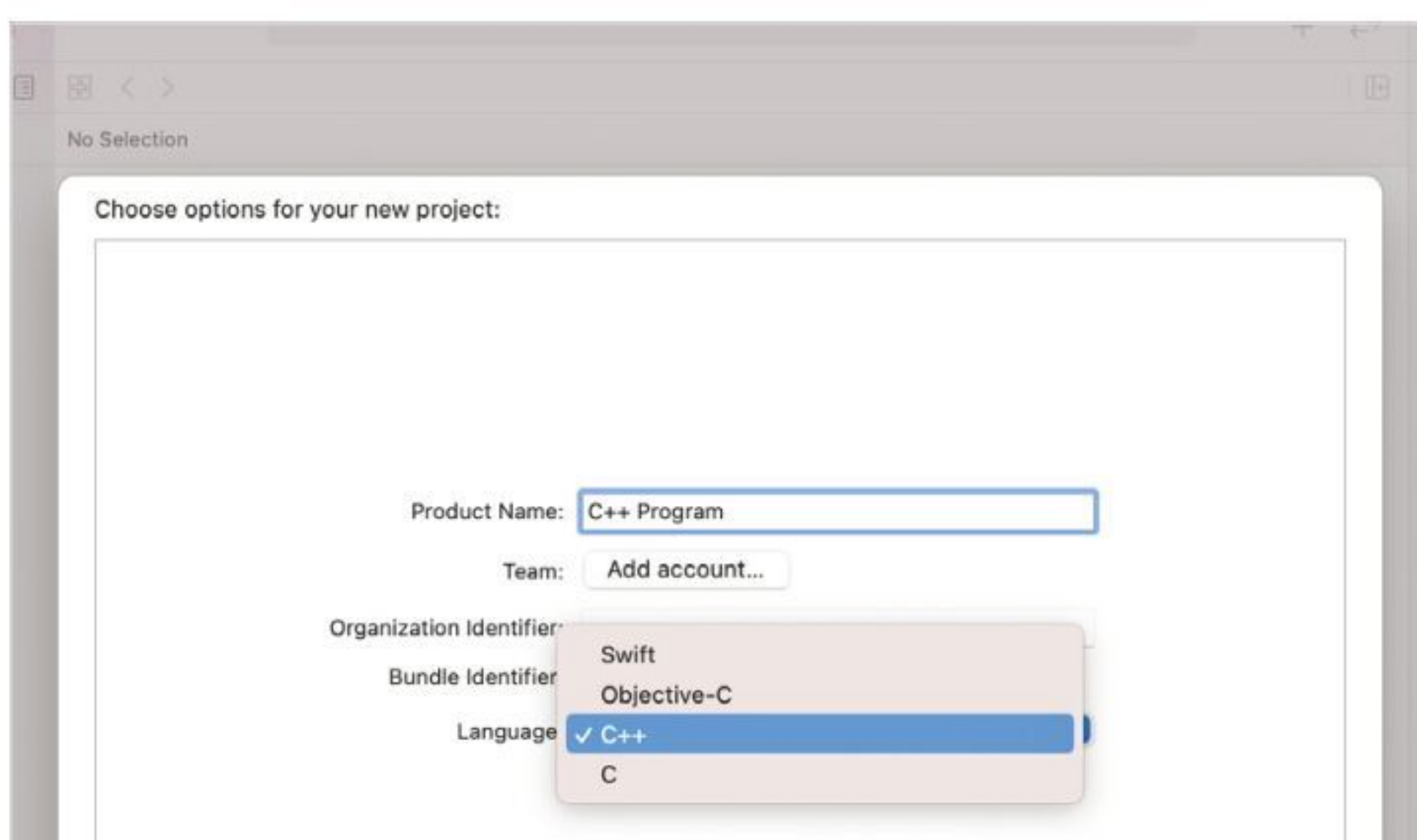
With everything now installed, including the additional components, Xcode will launch displaying the version number along with three choices and any recent projects that you've worked on – although for a fresh install, this will be blank.

**STEP 6**

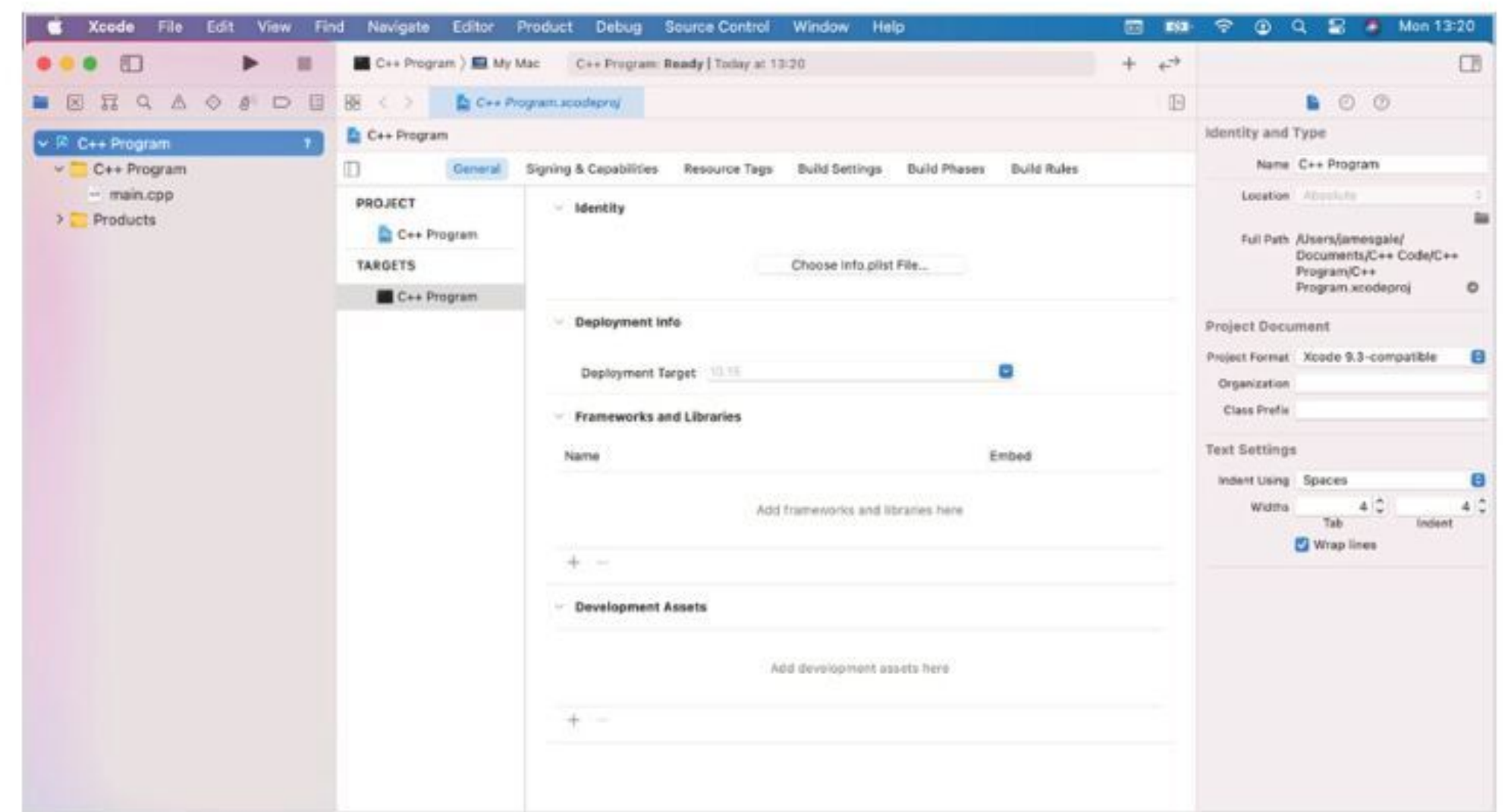
Start by clicking on Create New Xcode Project, this opens a template window to choose which platform you're developing code for. Click the macOS tab, then click the Command Line Tool option. Click Next to continue.

**STEP 7**

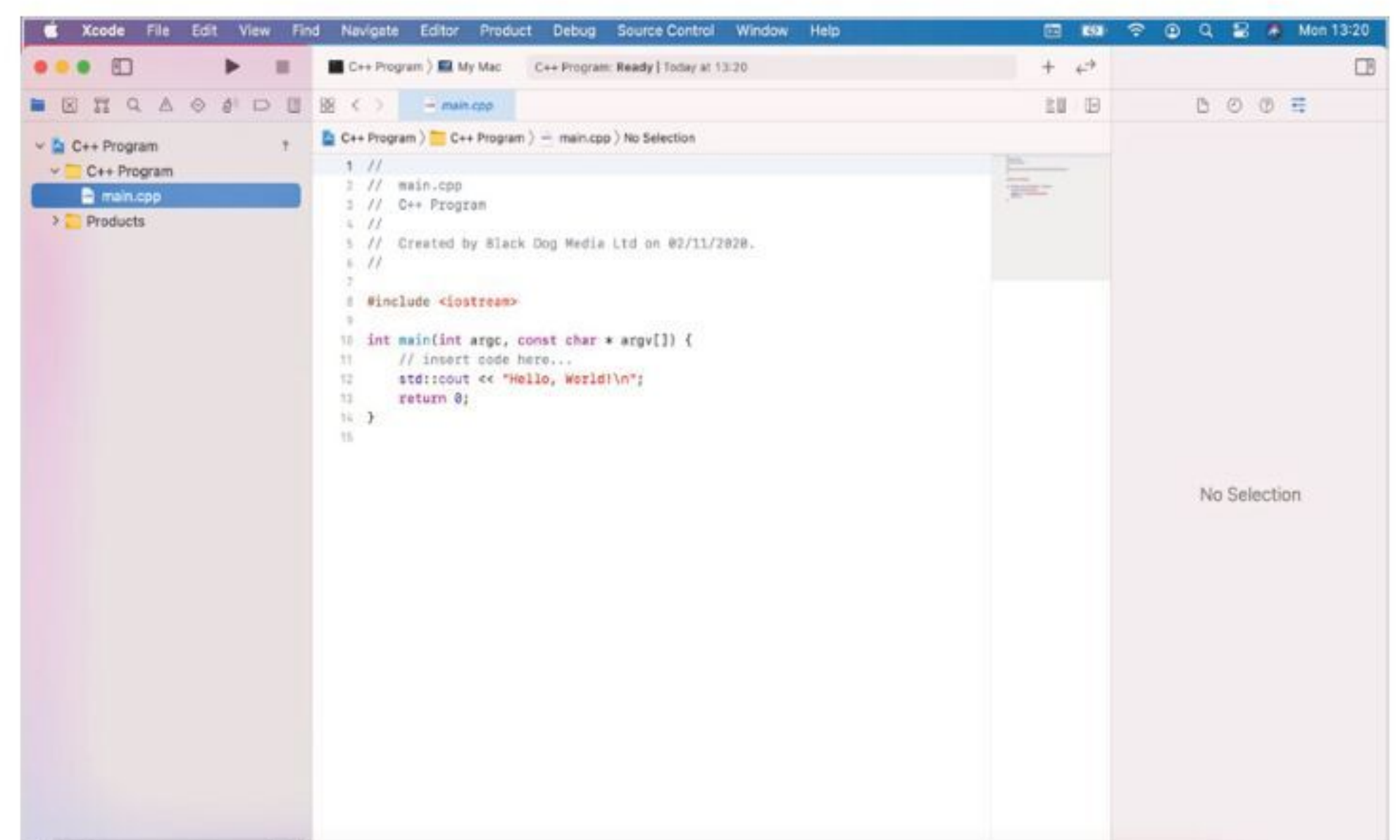
Fill in all the fields, but ensure that the Language option at the bottom is set to C++. Simply choose it from the drop-down list. When you've filled in the fields, and made sure that C++ is the chosen language, click on the Next button to continue.

**STEP 8**

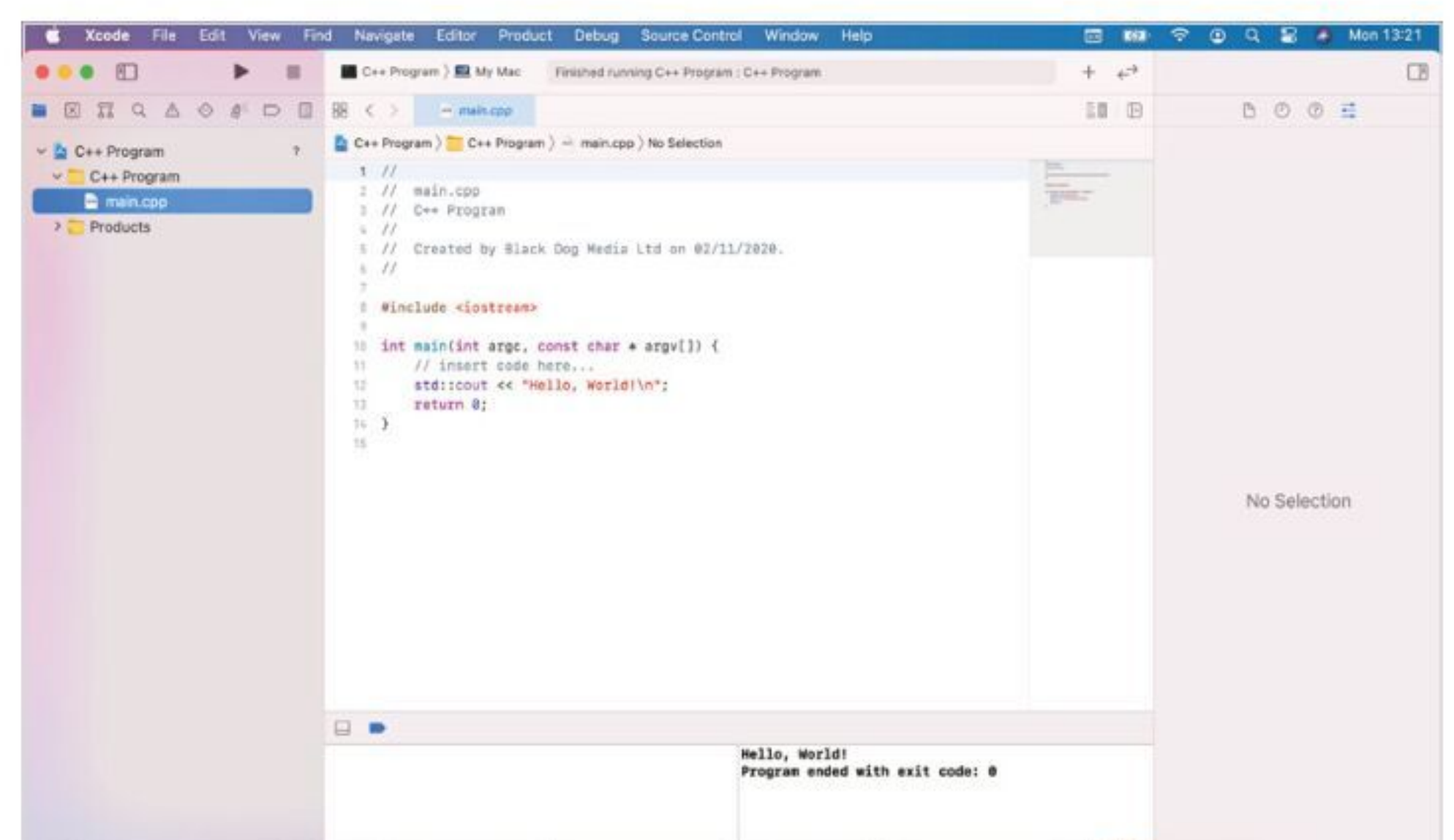
The next step asks where to create a Git Repository for all your future code. Choose a location on your Mac, or a network location, and click the Create button. When you've done all that, you can start to code. The left-hand pane details the files used in the C++ program you're coding. Click on the main.cpp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

**STEP 9**

You will notice 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 will be displayed at the bottom of the Xcode window.





How to Set Up C++ in Linux

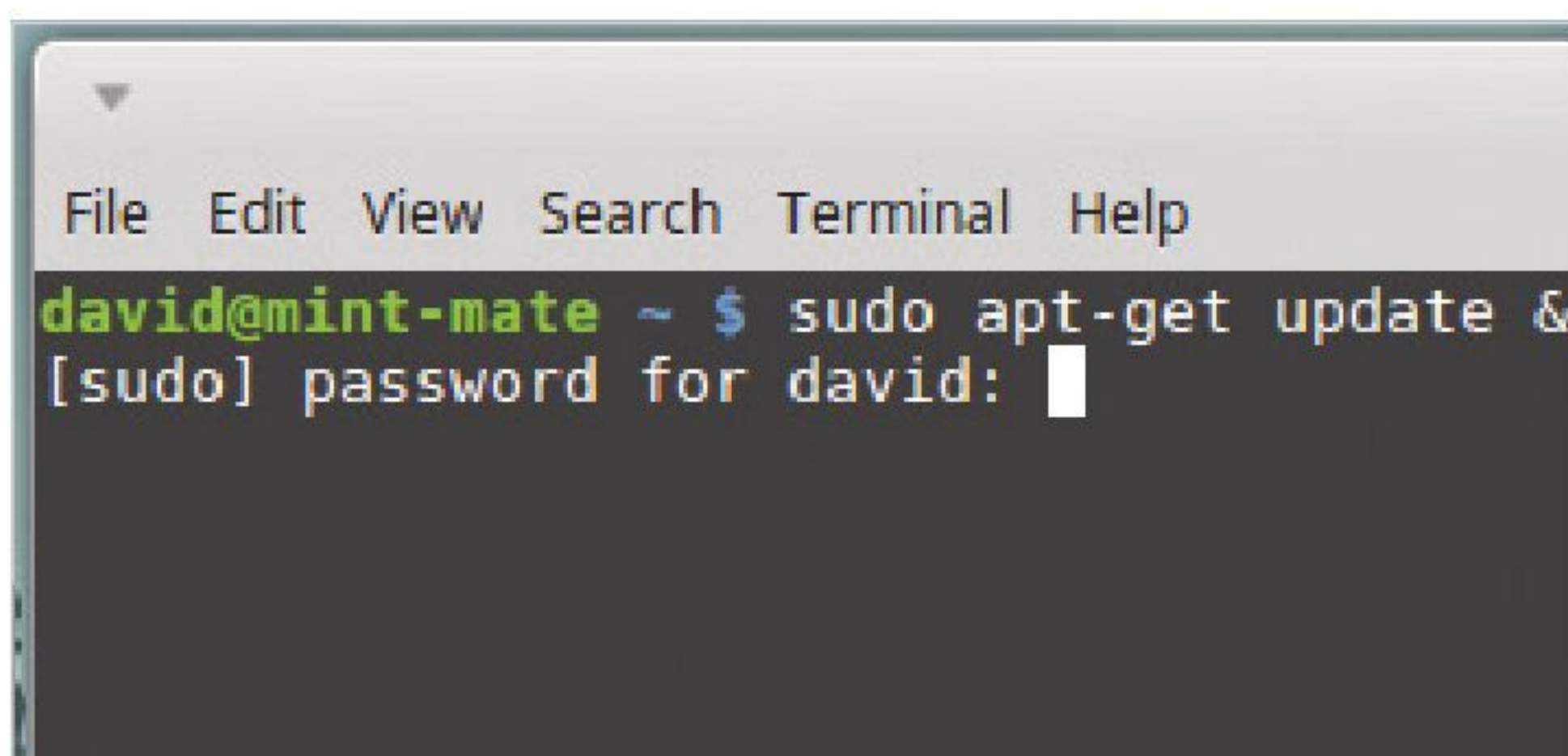
Linux is a great C++ coding environment. Most Linux distros already have the essential components preinstalled, such as a compiler and the text editors are excellent for entering code into, including colour coding; there's also tons of extra software available to help you out.

LINUX++

There are many different versions of Linux available, for this example we're using one of the more popular distributions: Linux Mint. However, these steps will work in any Debian-based Linux.

STEP 1

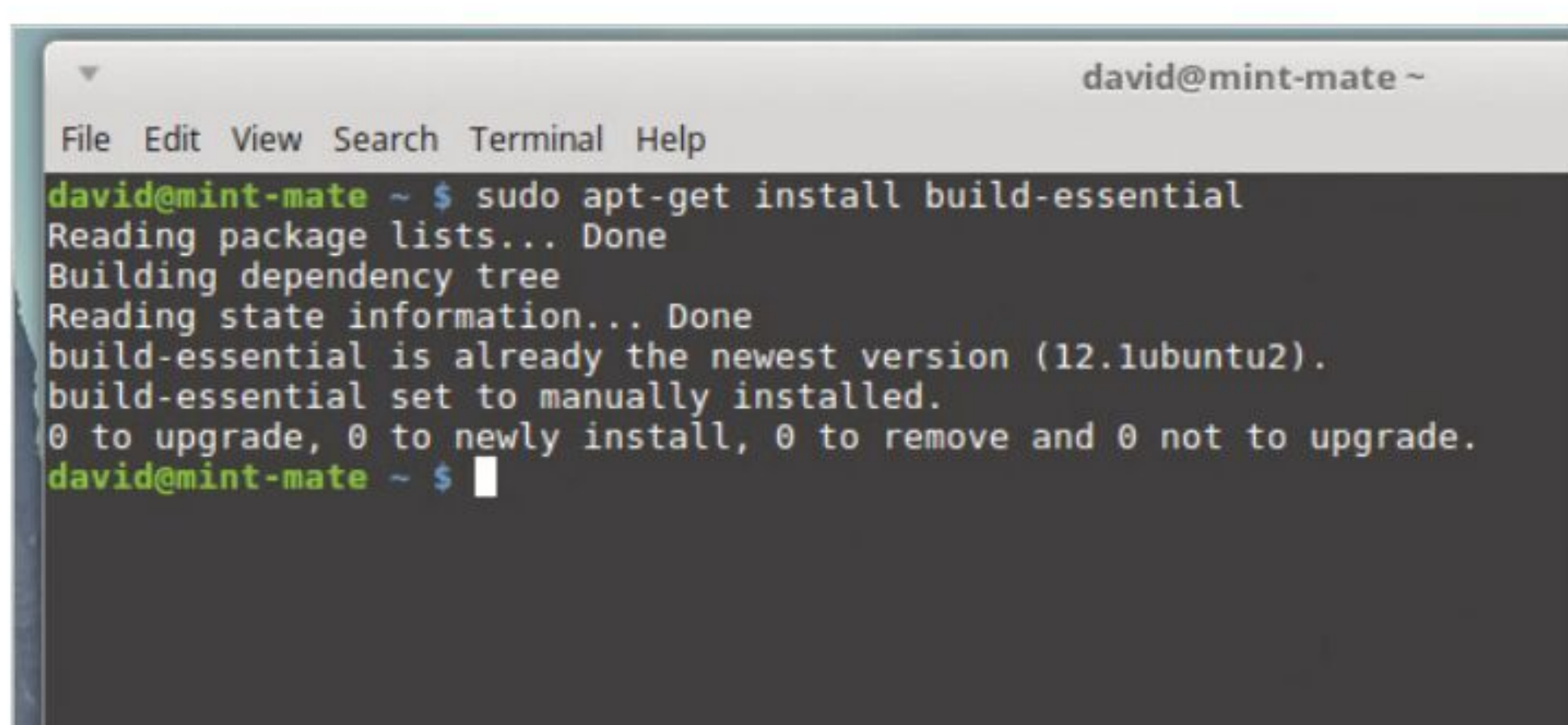
The first step with ensuring Linux is ready for your C++ code is check the system and software are up to date. Open a Terminal and enter: `sudo apt-get update && sudo apt-get upgrade`. Press Return and enter your password. These commands updates the entire system and any installed software.



```
File Edit View Search Terminal Help
david@mint-mate ~ $ sudo apt-get update &&
[sudo] password for david:
```

STEP 2

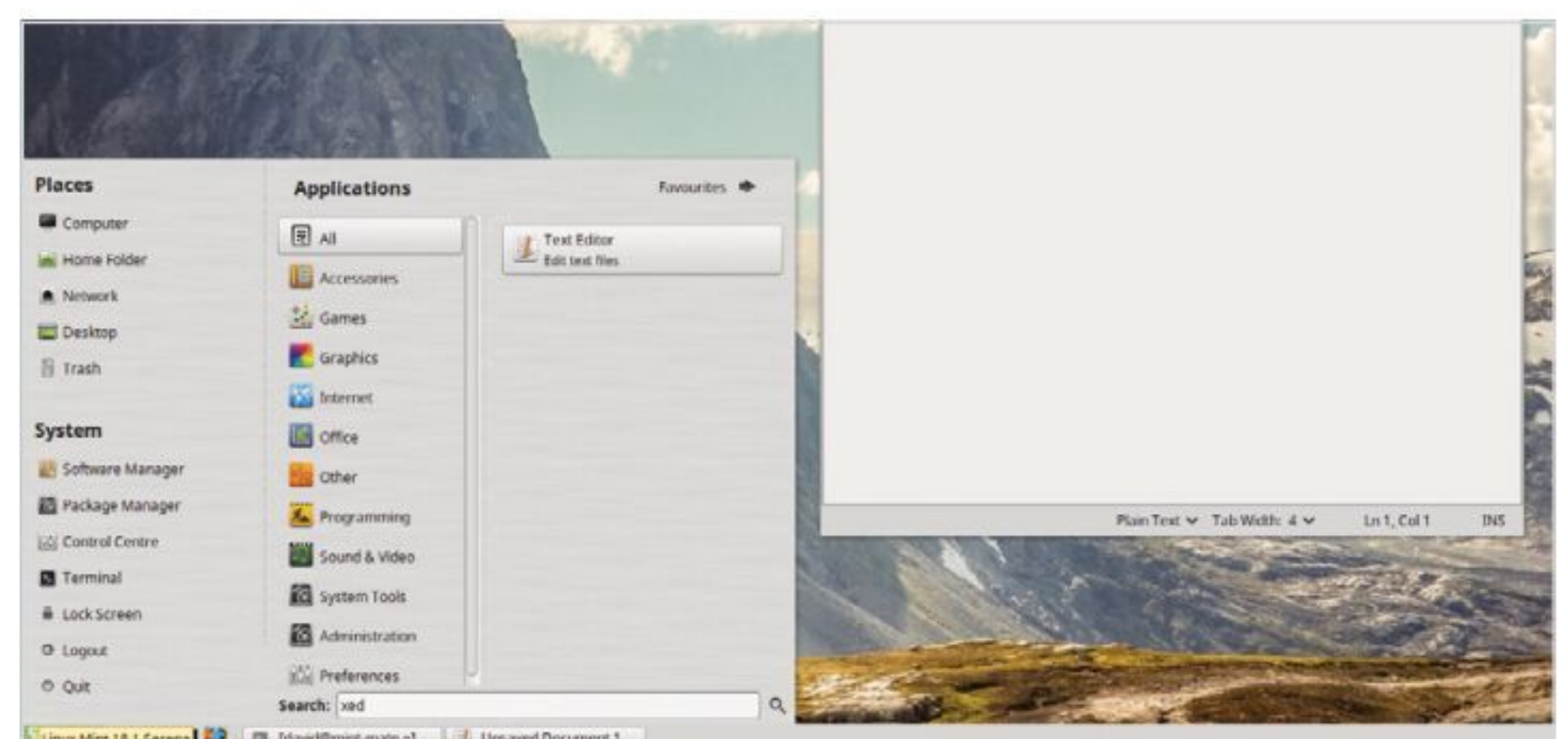
Most Linux distros come preinstalled with all the necessary components to start coding in C++. However, it's always worth checking to see if everything is present, so still within the Terminal, enter: `sudo apt-get install build-essential` and press Return. If you have the right components, nothing is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



```
File Edit View Search Terminal Help
david@mint-mate ~ $ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
0 to upgrade, 0 to newly install, 0 to remove and 0 not to upgrade.
david@mint-mate ~ $
```

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.

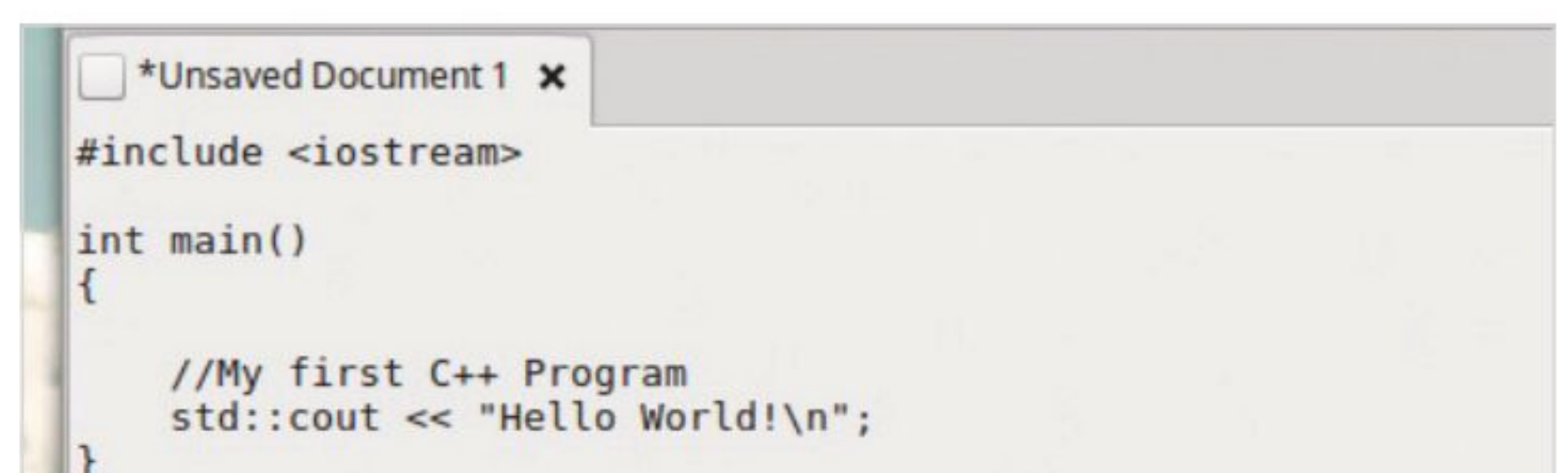


STEP 4

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, it's:

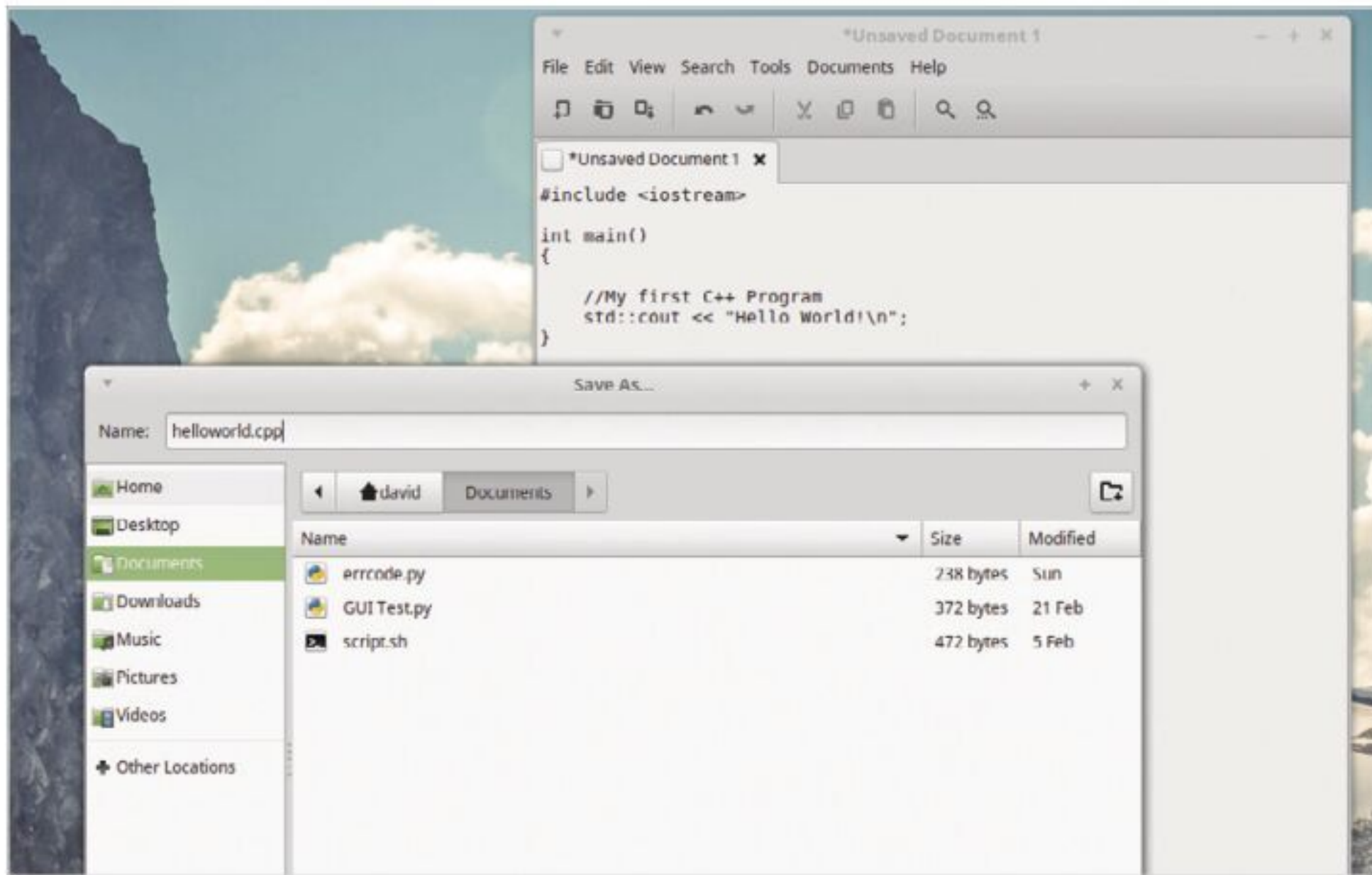
```
#include <iostream>

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

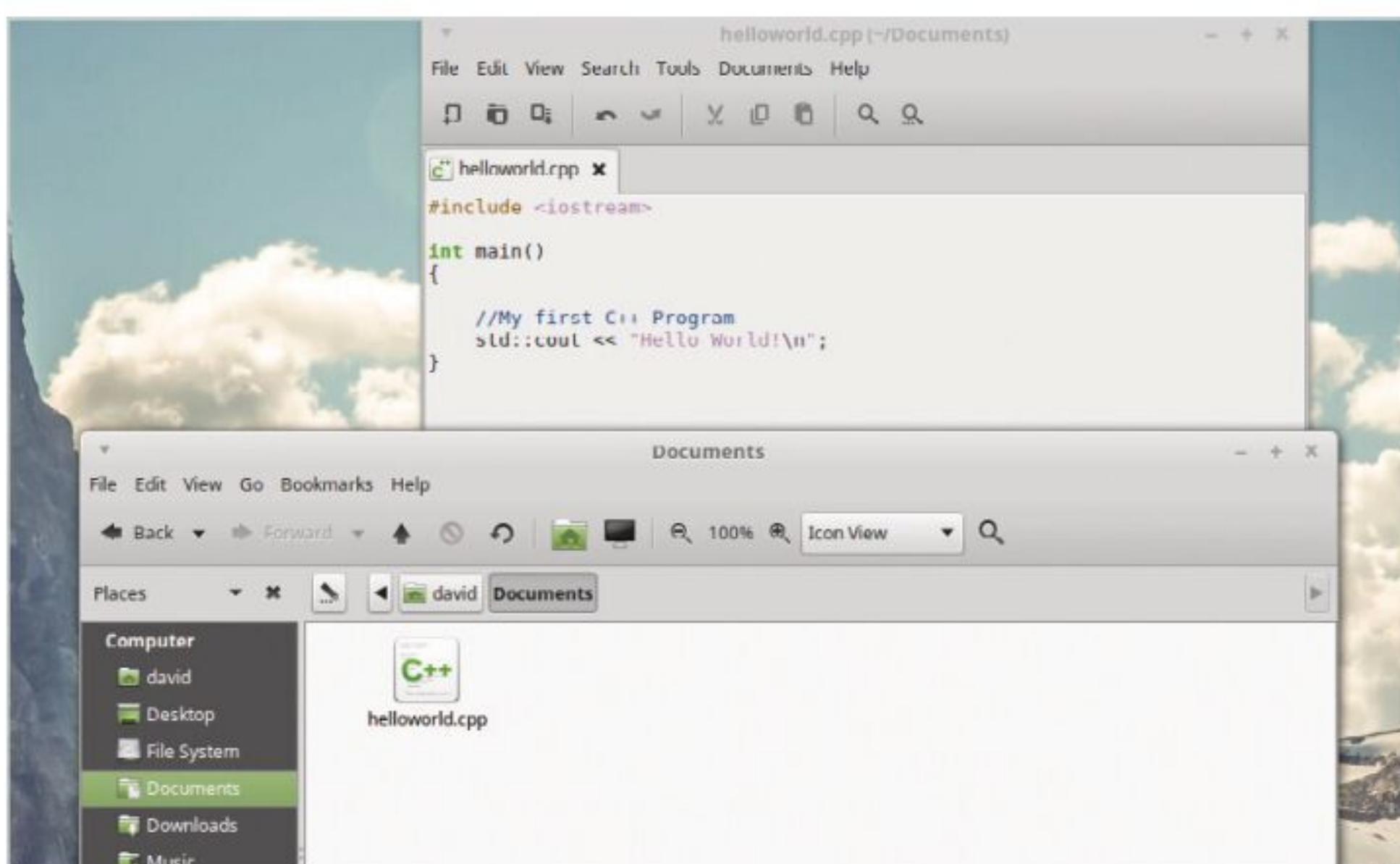


STEP 5

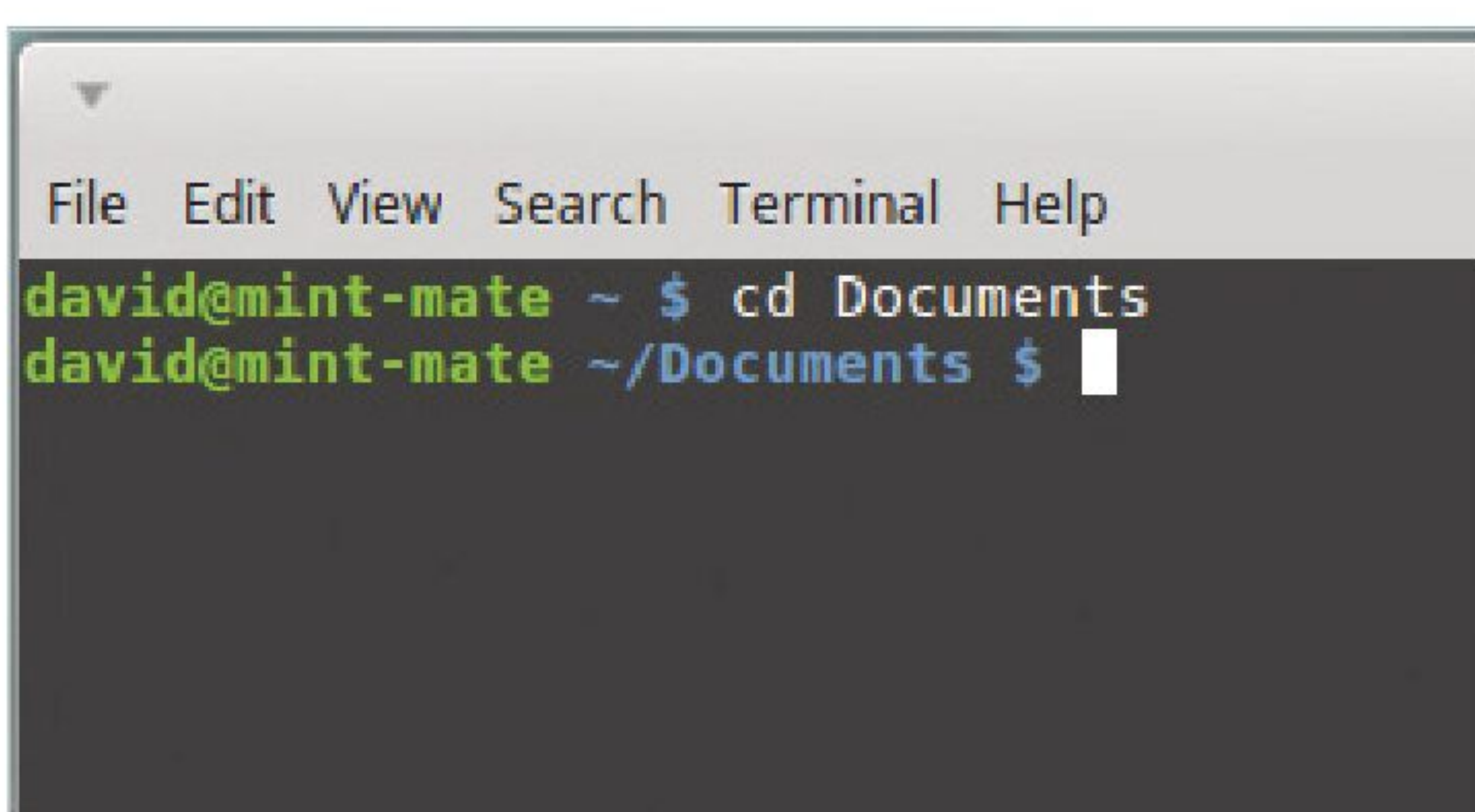
When you've entered your code, click File > Save As and choose a folder where you want to save your program. Name the file as helloworld.cpp, or any other name just as long as it has .cpp as the extension. Click Save to continue.

**STEP 6**

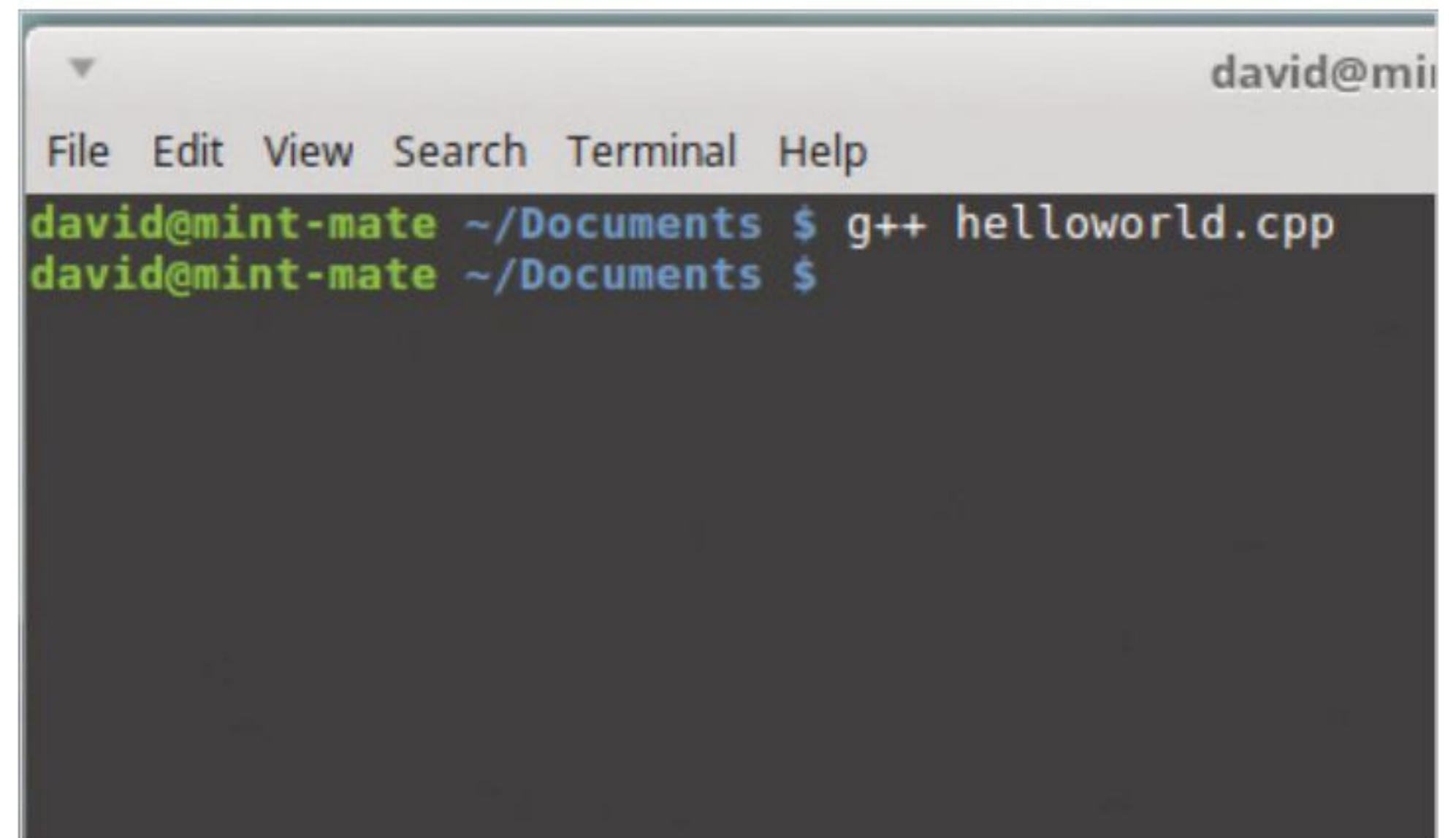
The first thing you can see is that Xed has automatically recognised this as a C++ file, since the file extension is now set to .cpp. The colour coding is present in the code and if you open up the file manager you can also see that the file's icon has C++ stamped on it.

**STEP 7**

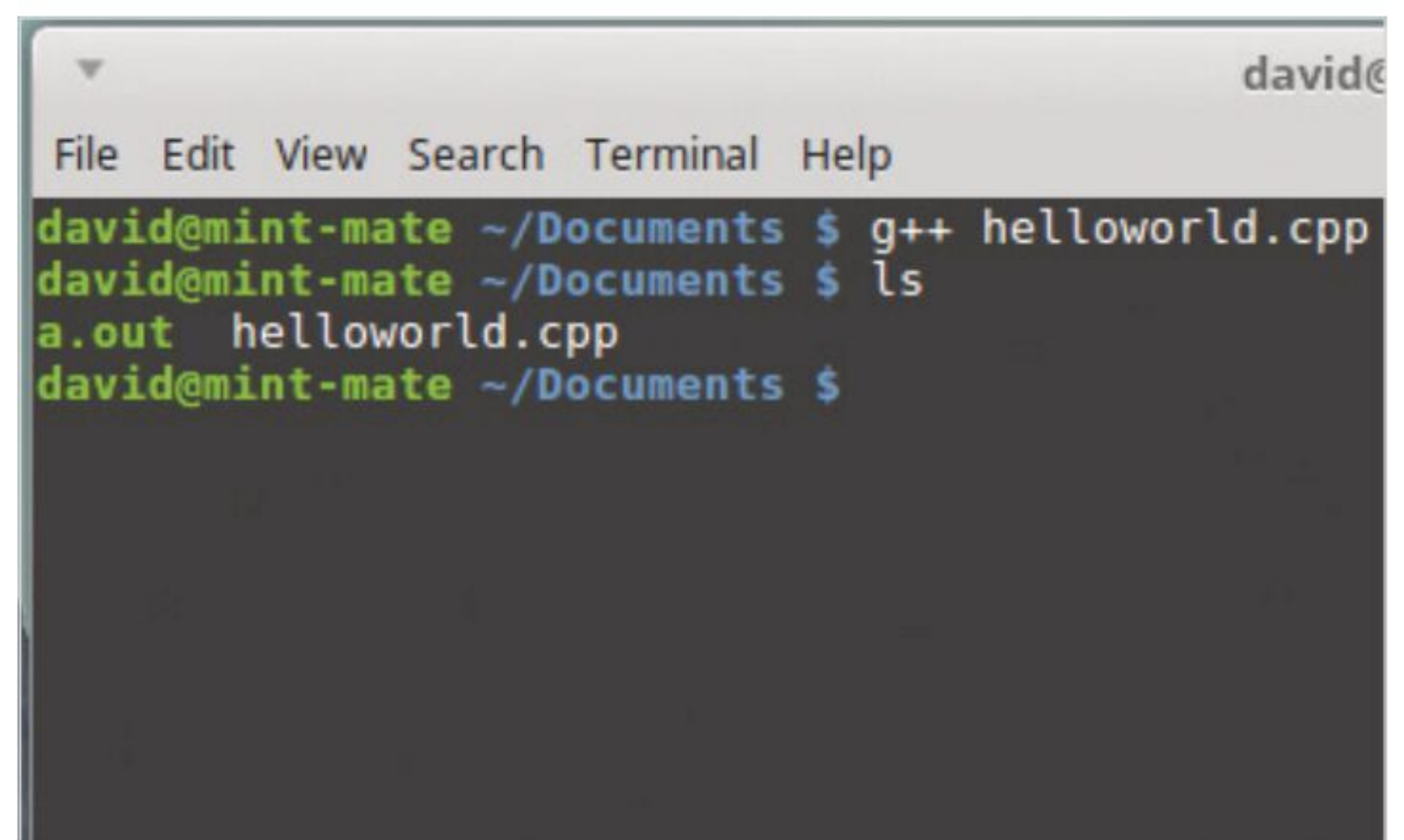
With your code now saved, drop into the Terminal again. You need to navigate to the location of the C++ file you've just saved. Our example is in the Documents folder, so we can navigate to it by entering: `cd Documents`. Remember, the Linux Terminal is case sensitive, so any capitals must be entered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

**STEP 8**

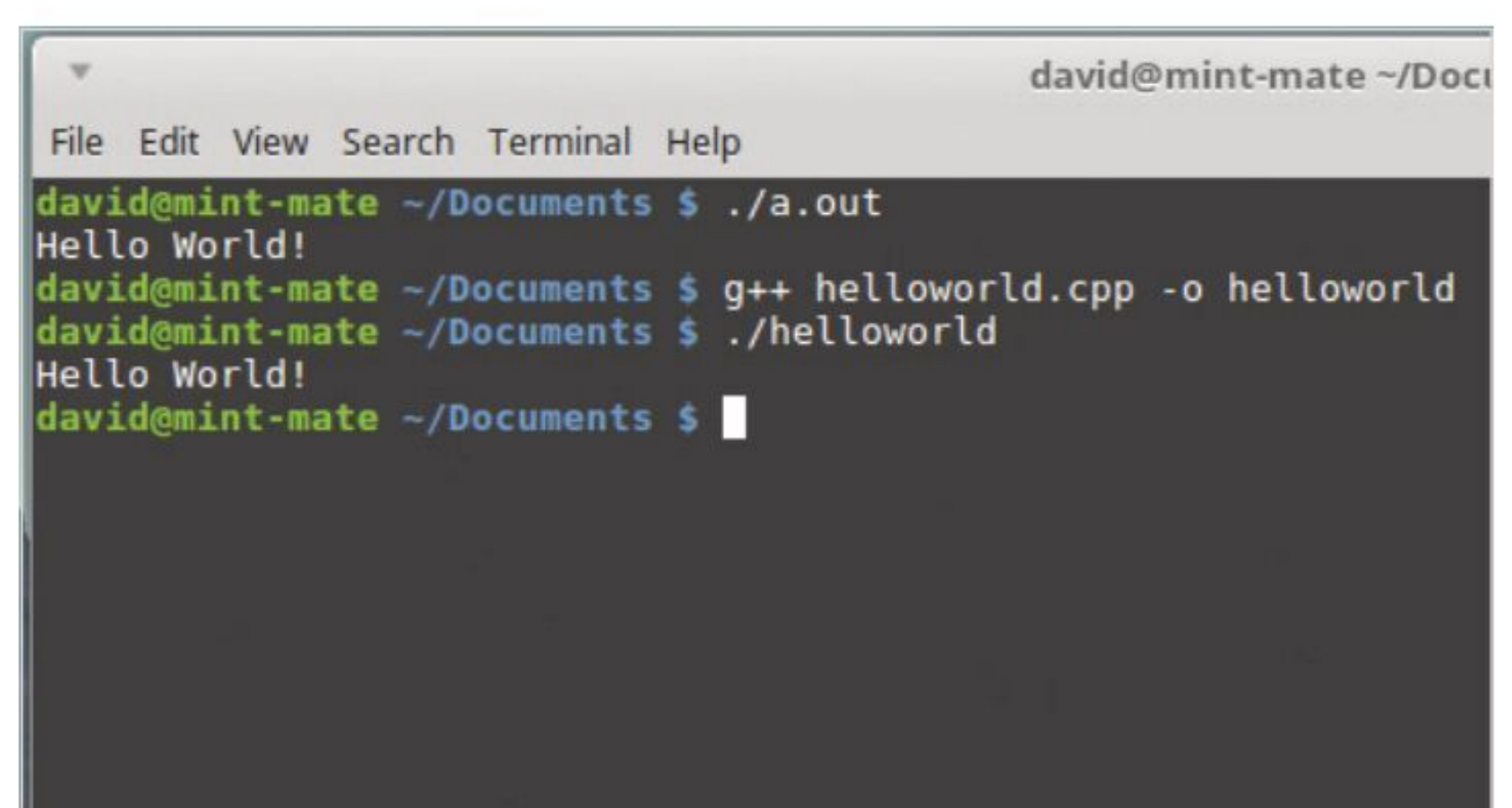
Before you can execute the C++ file you need to compile it. In Linux it's common to use g++, an open source C++ compiler and as you're now in the same folder as the C++ file, go to the Terminal, enter: `g++ helloworld.cpp` and press return.

**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

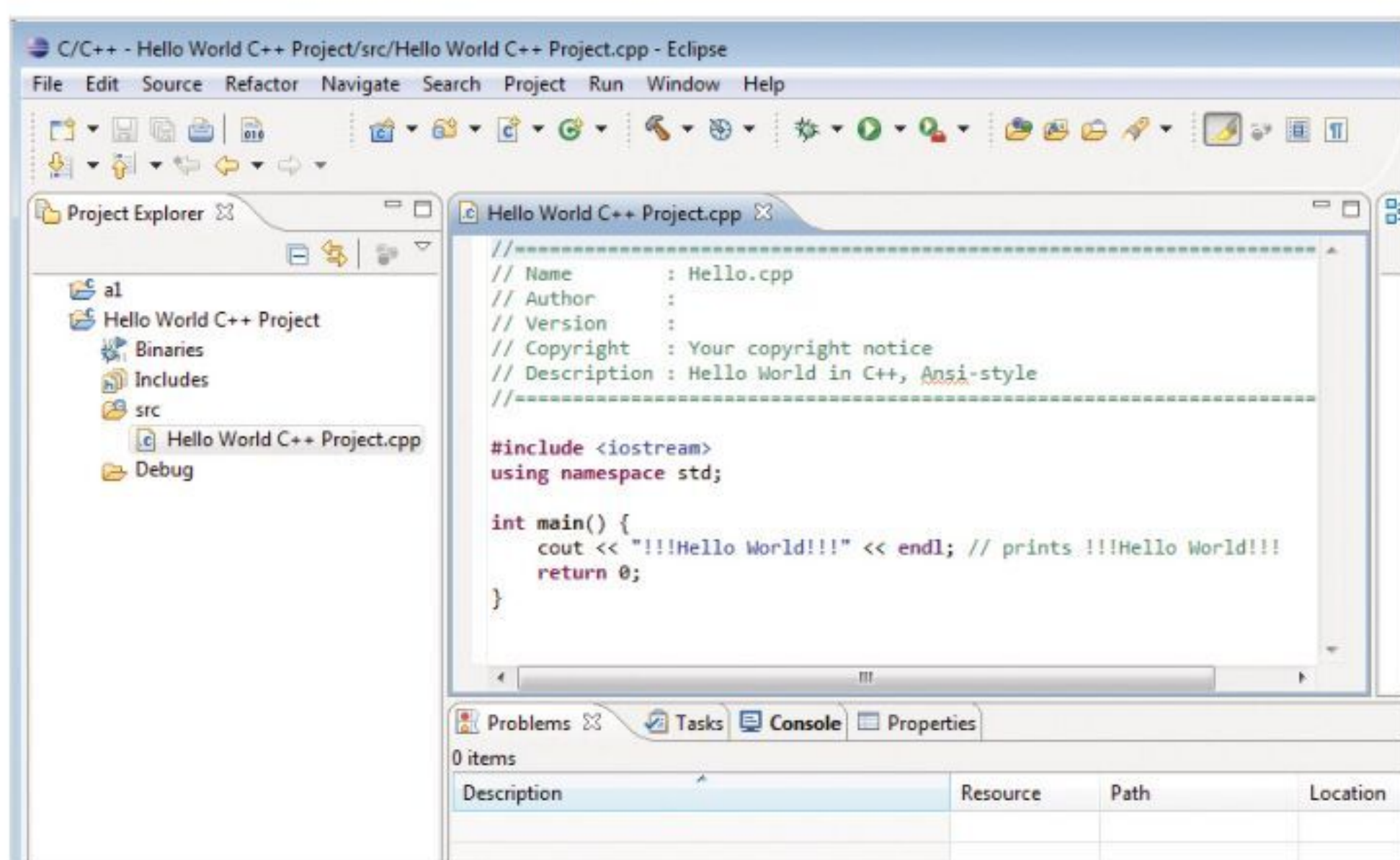
If you want to try a different approach to working with your C++ code, then there are plenty of options available to you. Windows is the most prolific platform for C++ IDEs but there are plenty for Mac and Linux users too.

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.

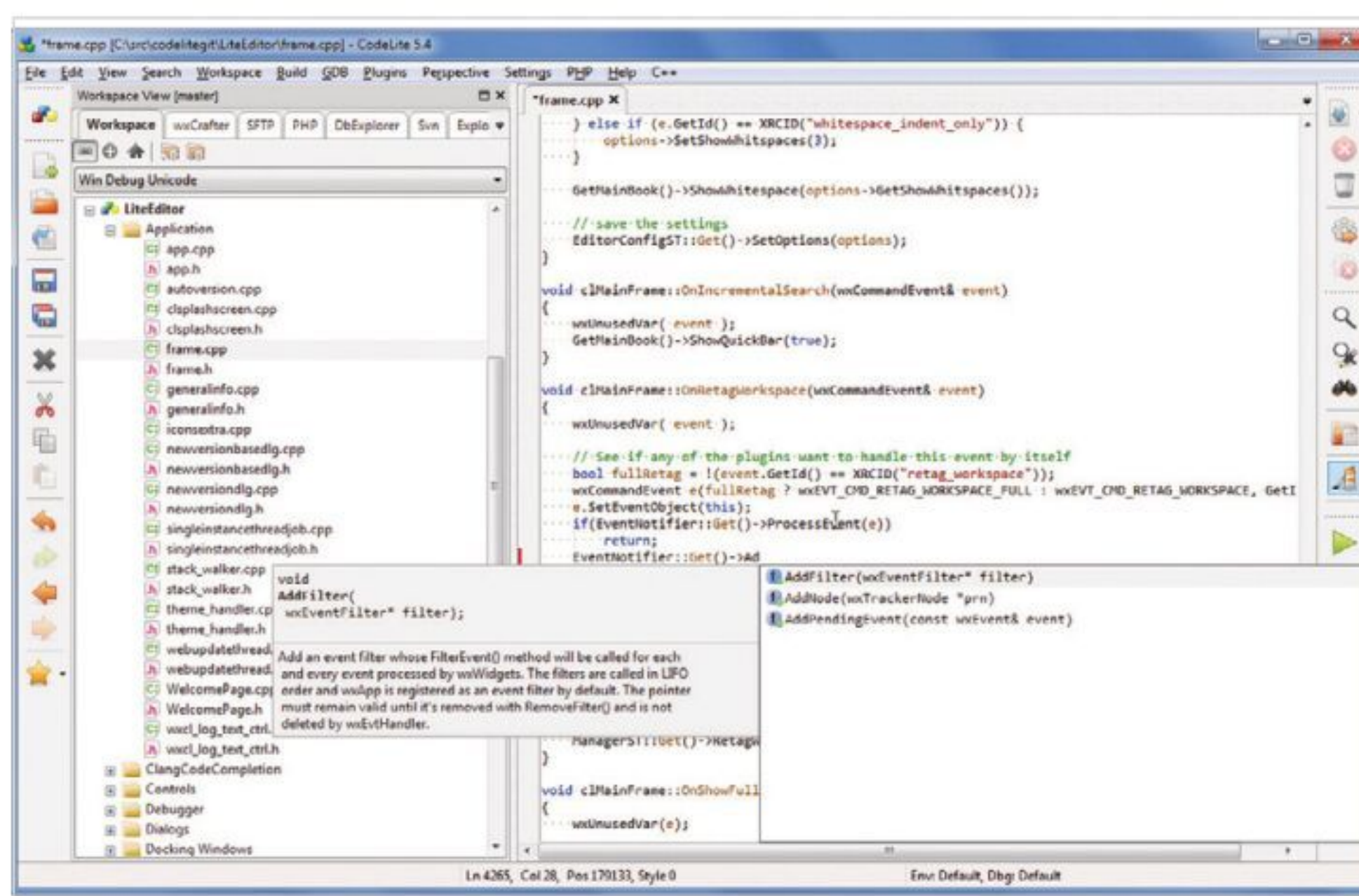
ECLIPSE

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.



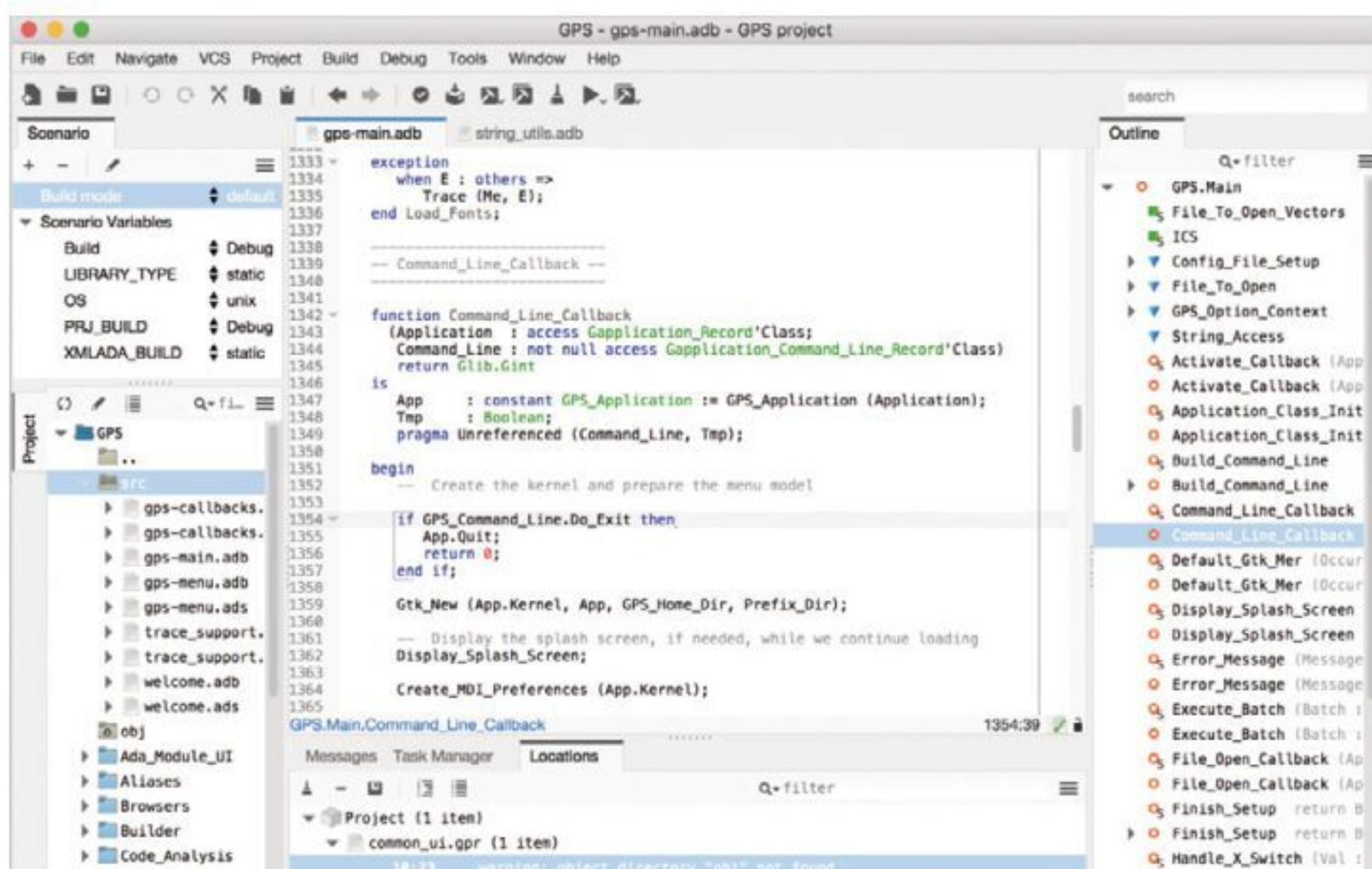
CODELITE

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/.



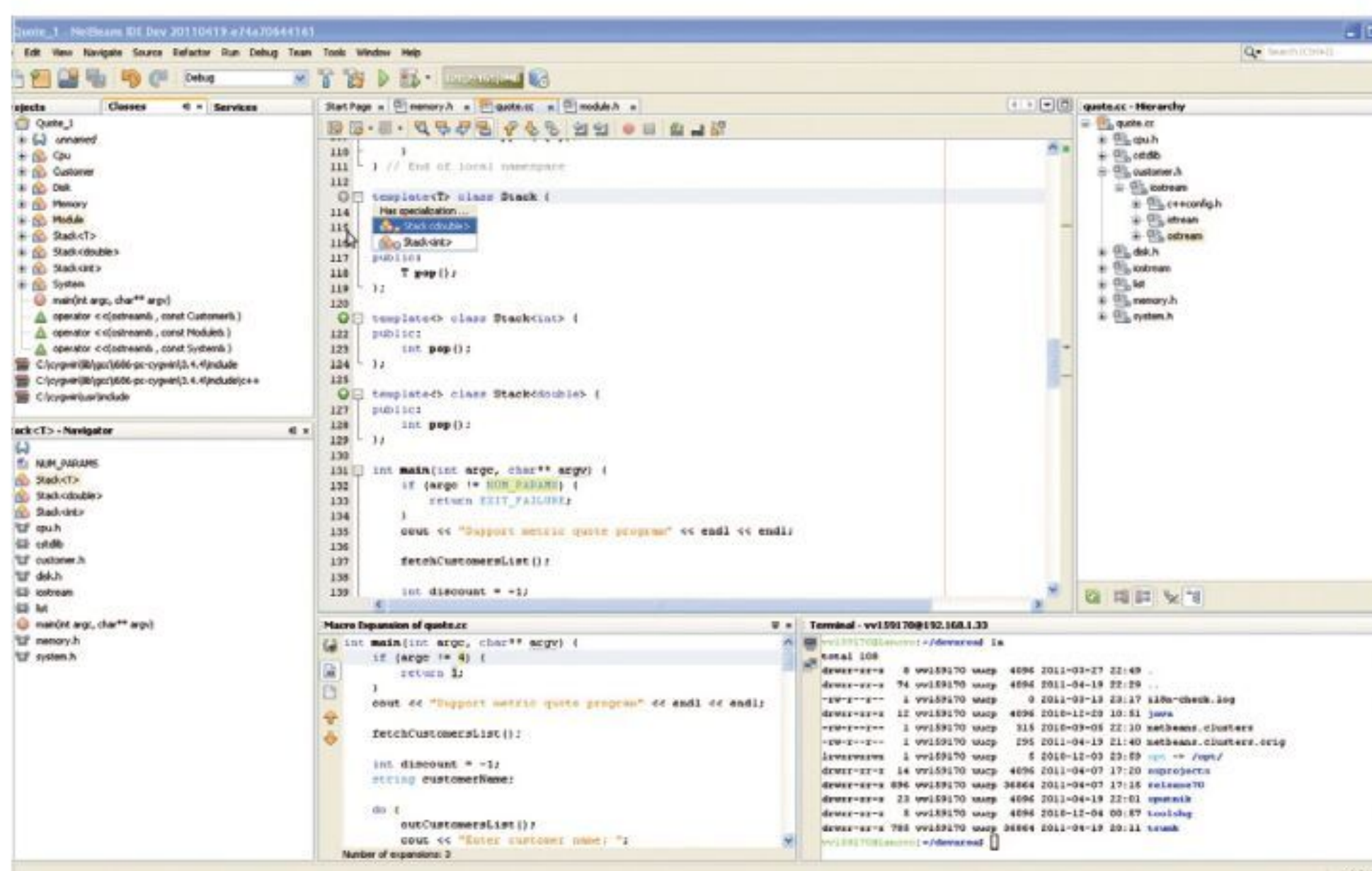
GNAT

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

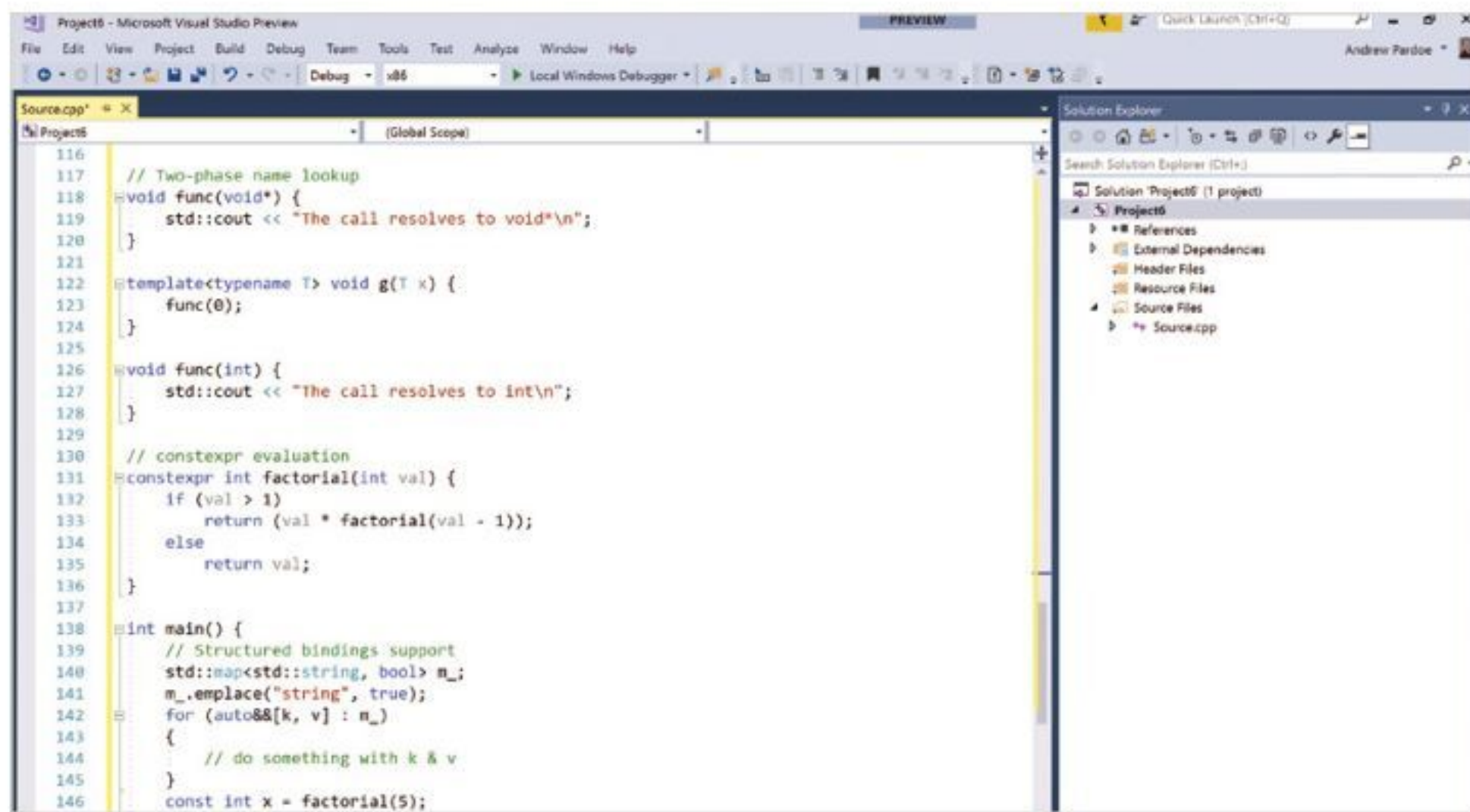
Another popular choice is NetBeans. This is another excellent IDE that's packed with features and a pleasure to use. NetBeans IDE includes project based templates for C++ that give you the ability to build applications with dynamic and static libraries. Find out more at www.netbeans.org/features/cpp/index.html.





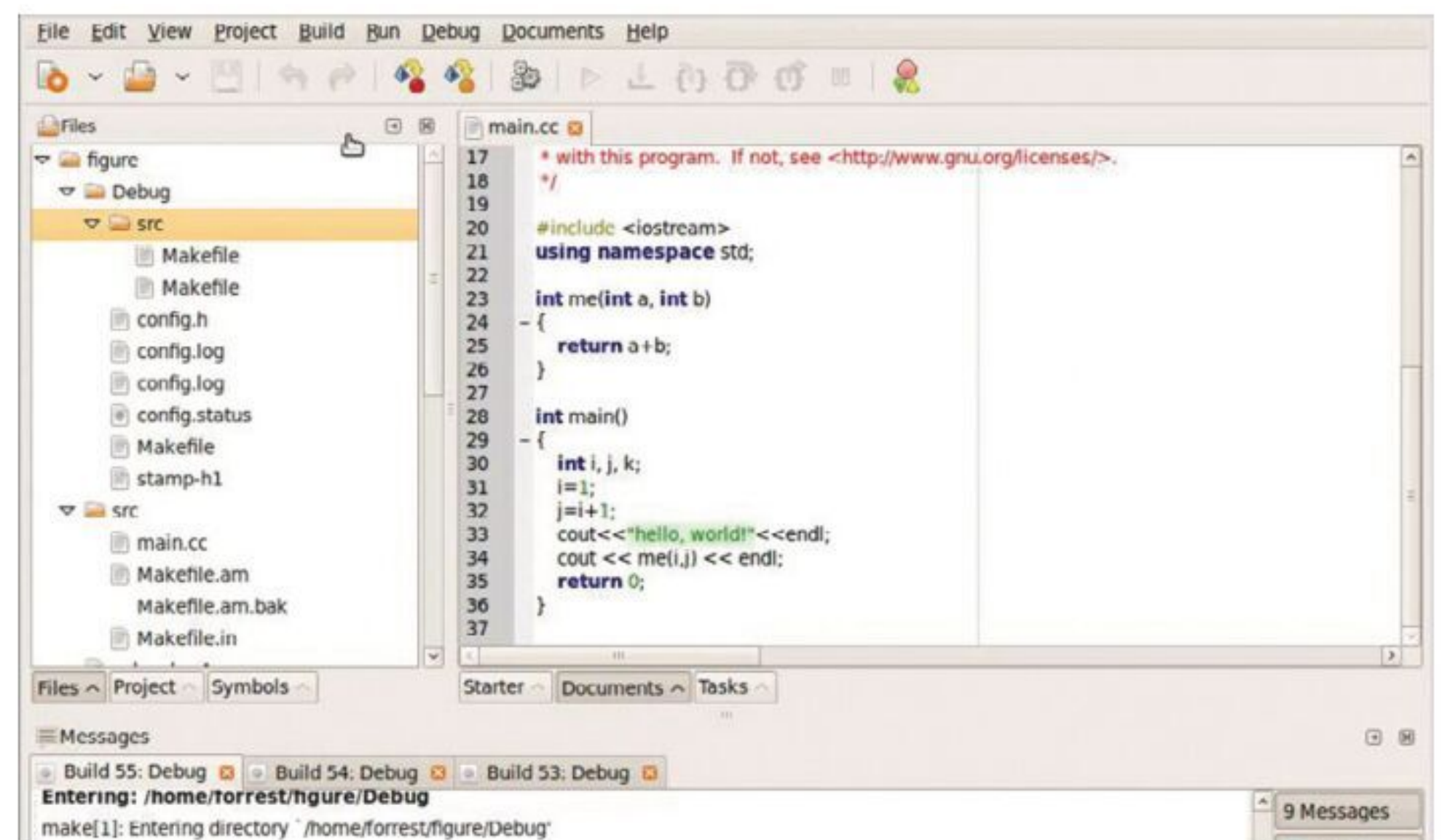
VISUAL STUDIO

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.



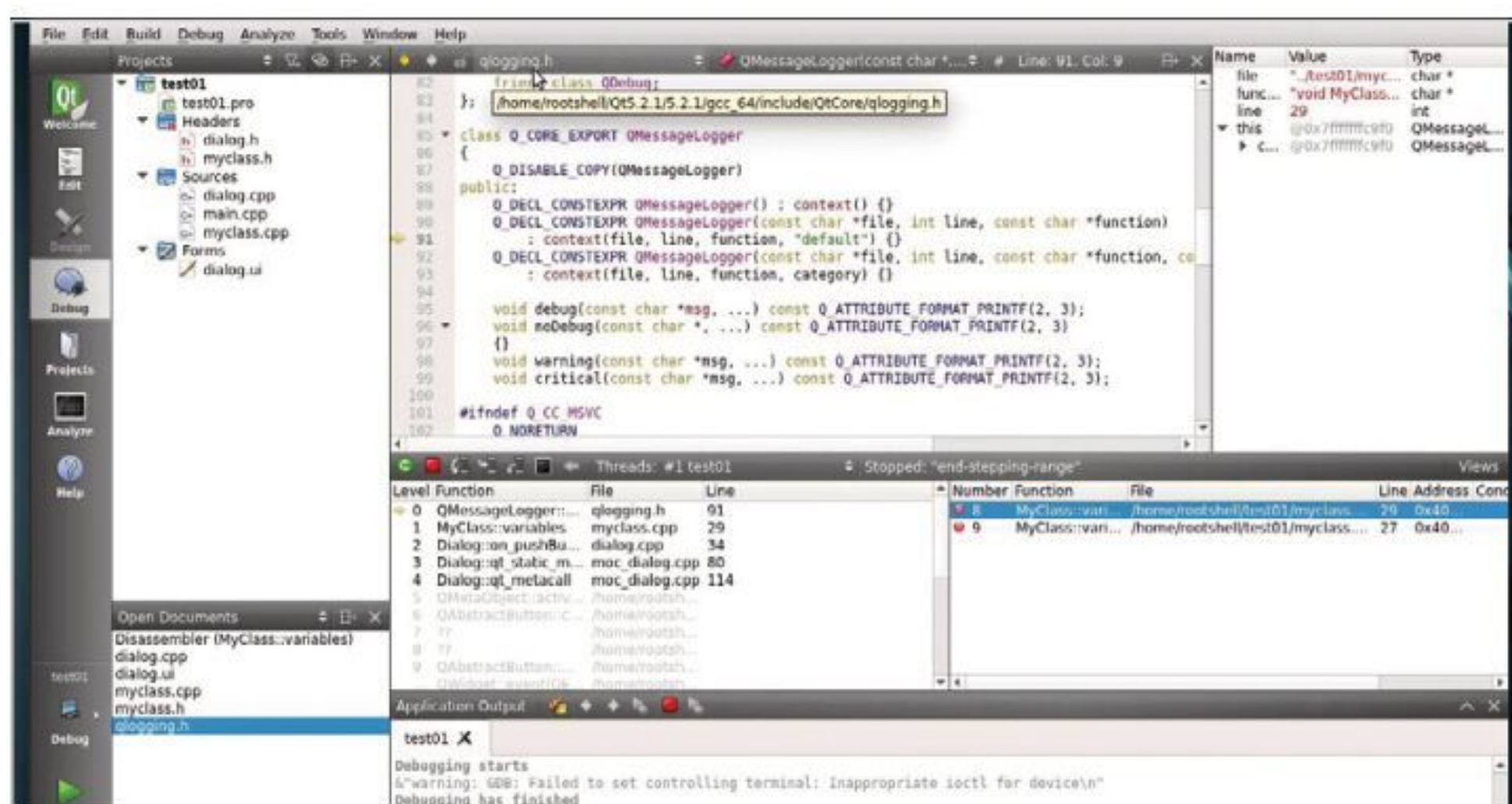
ANJUTA

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.



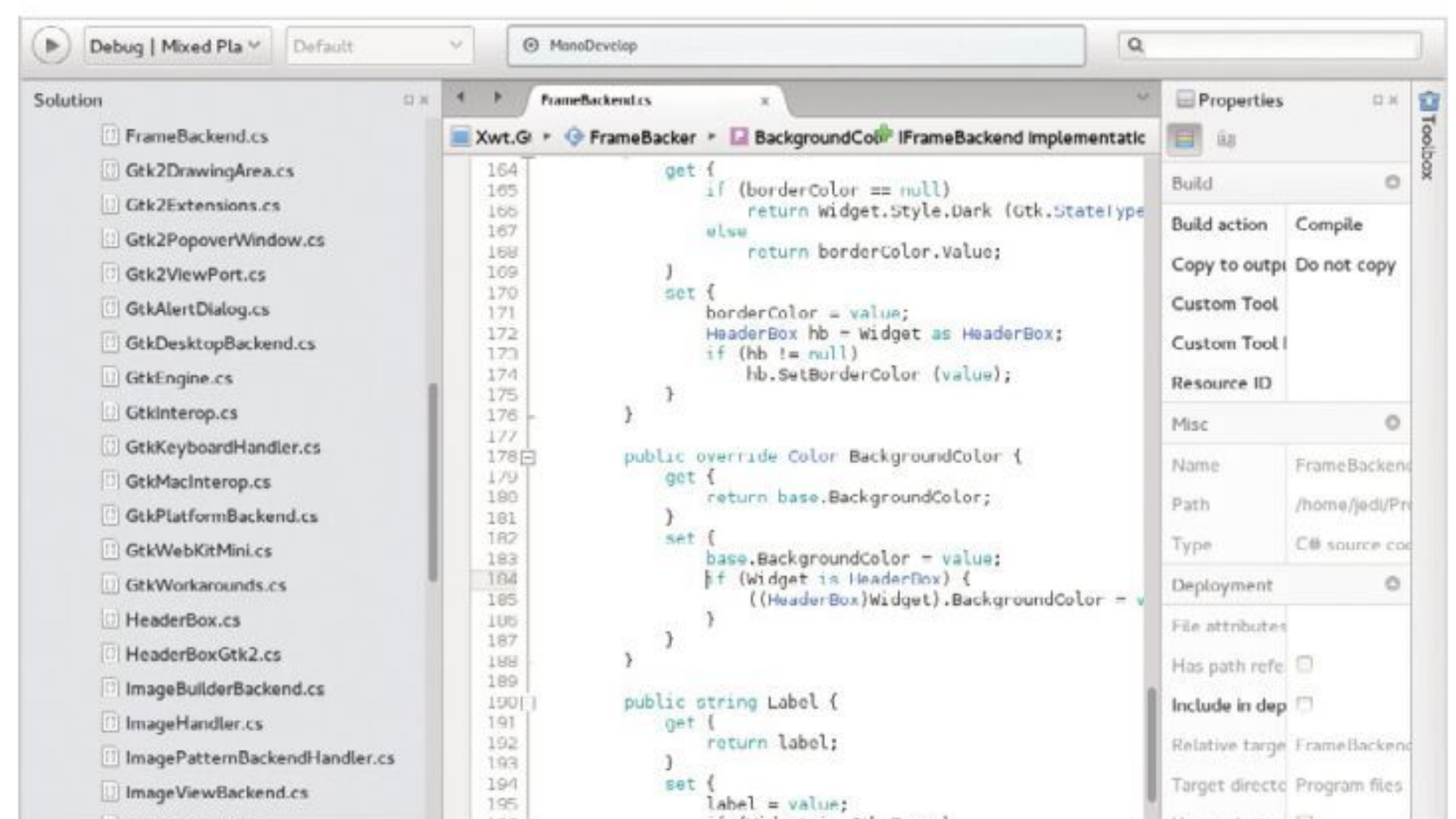
QT CREATOR

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 your 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/.



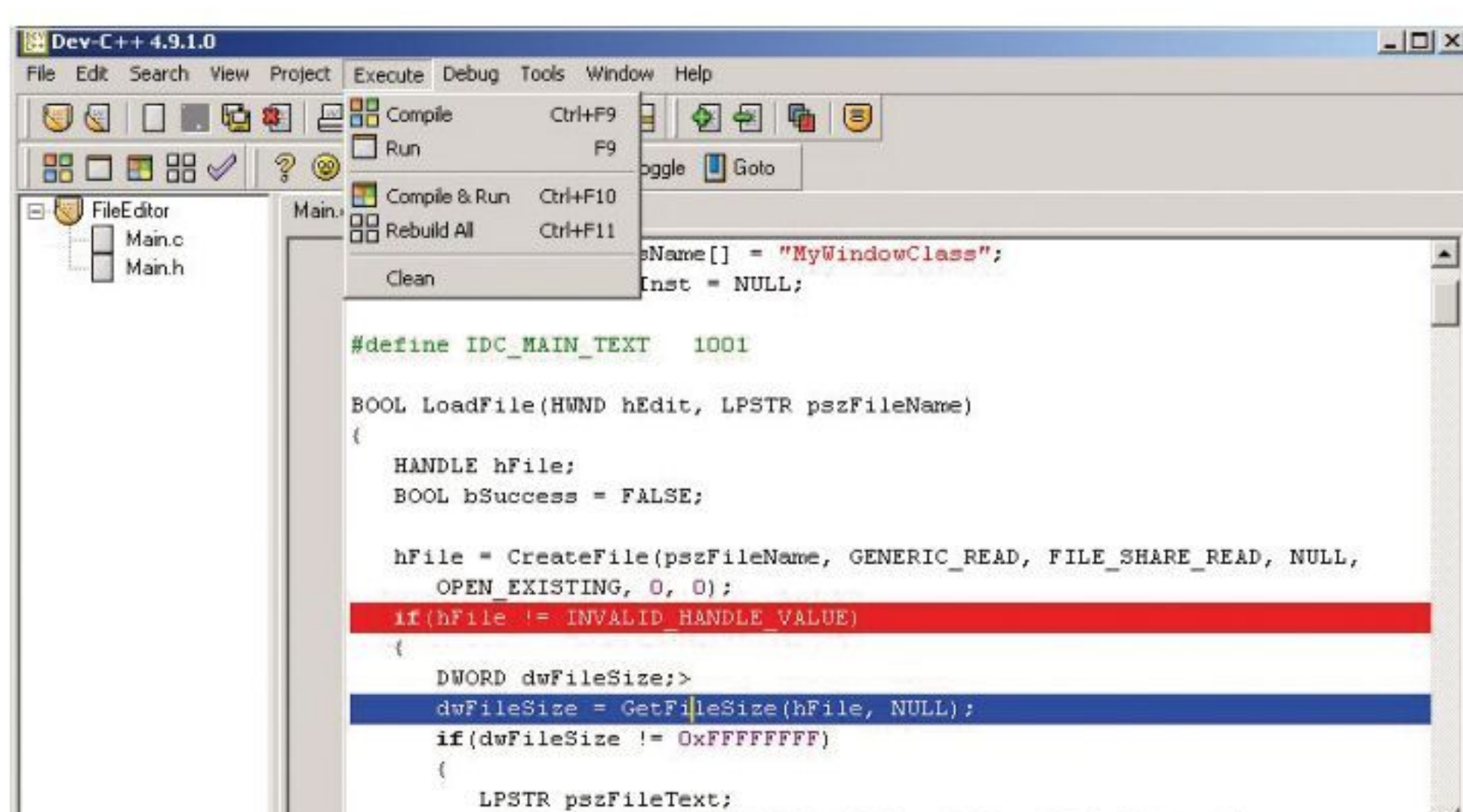
MONODEVELOP

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/.



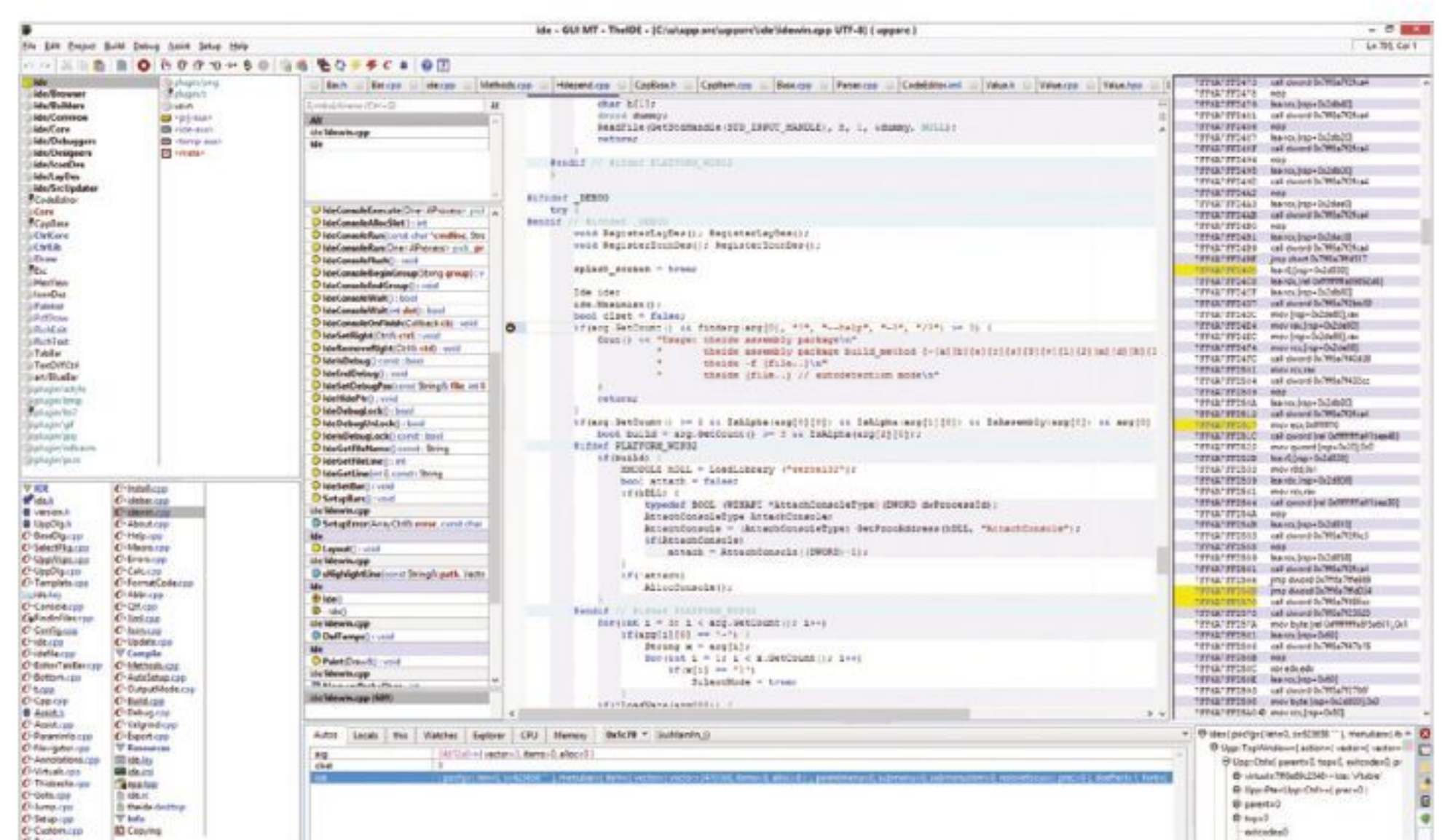
DEV C++

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 boasts 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.





Coding with C++

This section contains the building block of C++, from learning how to compile and execute your first C++ code, through to developing user interaction. There's a lot to learn with C++, but these tutorials, tips and tricks will lead you in the right direction.

C++ is different to the other programming languages in this book, and it's continually evolving as the applications that require it are forever improving. If you know C++, then you're well on your way to becoming a much sought-after coder and a vital member of the coding community.



Your First C++ Program

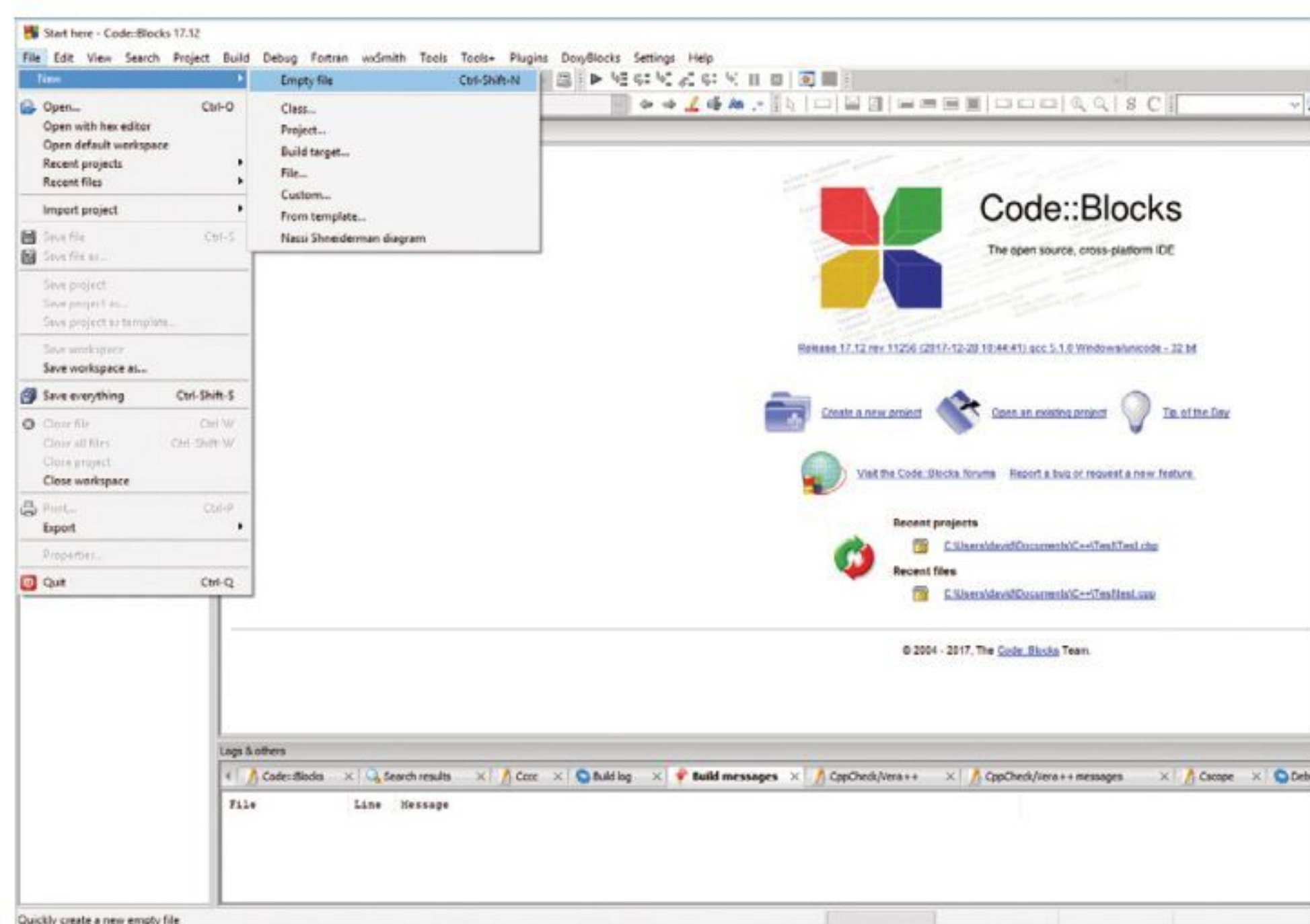
You may have followed the Mac and Linux examples previously but you're going to be working exclusively in Windows and Code::Blocks from here on. Let's begin by writing your first C++ program and taking the first small step into a larger coding world.

HELLO, WORLD!

It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

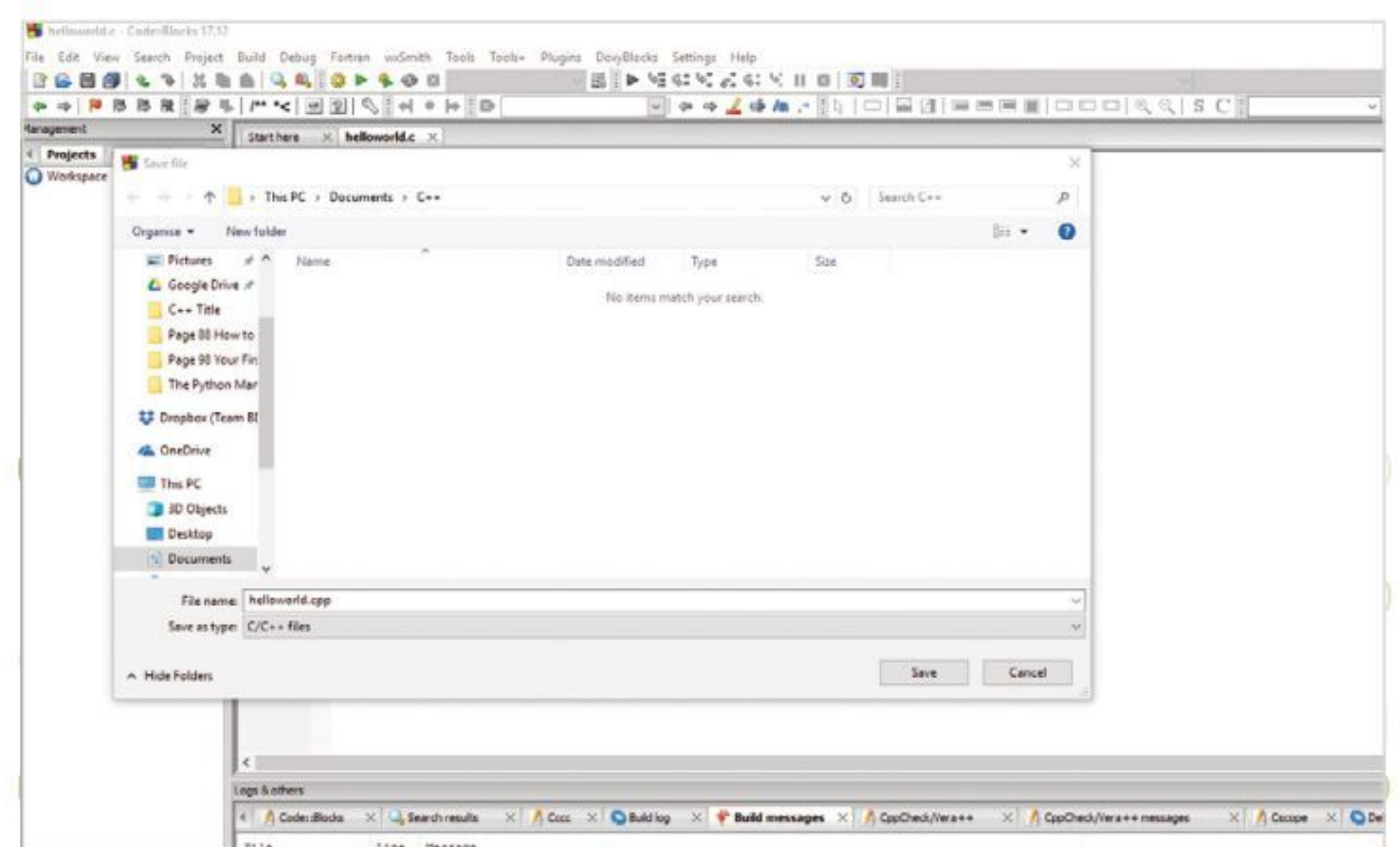
STEP 1

We're going to use Windows 10 and the excellent IDE Code::Blocks for the rest of the C++ code and tutorials in this book. Begin by launching Code::Blocks. When open, click on File > New > Empty File or press Ctrl+Shift+N on the keyboard.



STEP 3

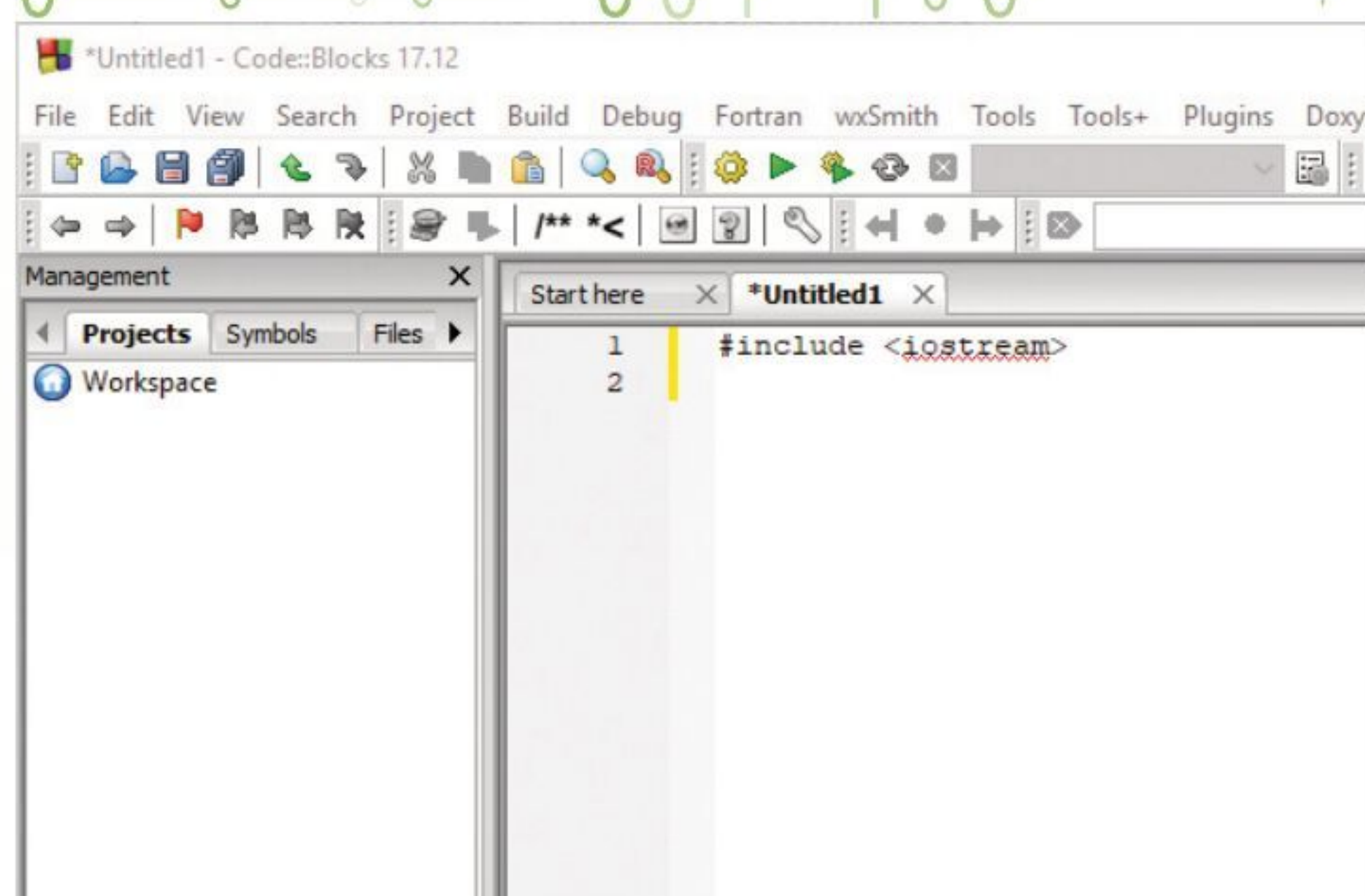
At the moment it doesn't look like much, and it makes even less sense, but we'll get to that in due course. Now click on File > Save File As. Create or find a suitable location on your hard drive and in the File Name box, call it helloworld.cpp. Click the Save as type box and select C/C++ files. Click the Save button.



STEP 2

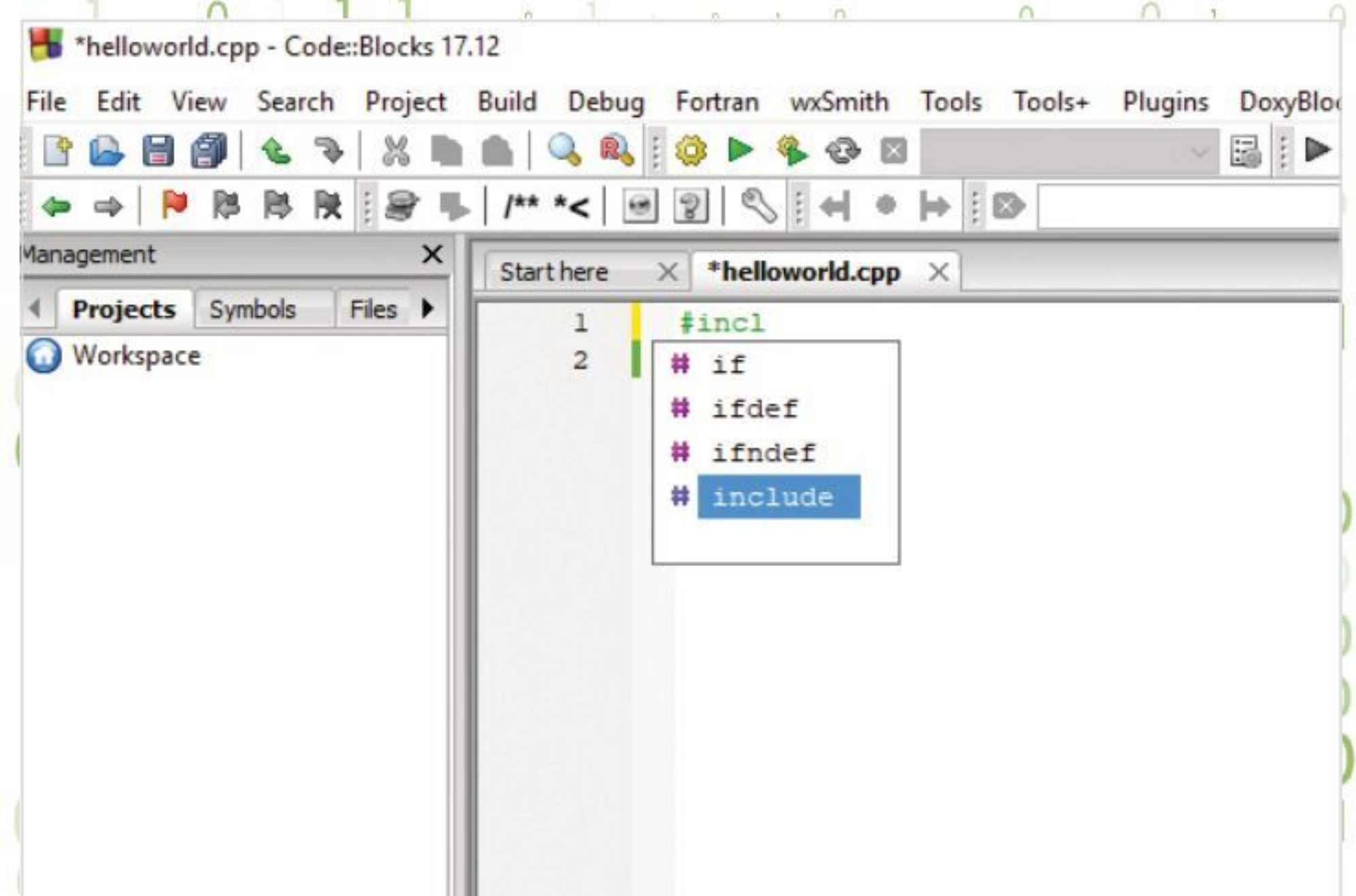
Now you can see a blank screen, with the tab labelled *Untitled1, and the number one in the top left of the main Code::Blocks window. Begin by clicking in the main window, so the cursor is next to the number one, and entering:

```
#include <iostream>
```



STEP 4

You can see that Code::Blocks has now changed the colour coding, recognising that the file is now C++ code. This means that code can be auto-selected from the Code::Blocks repository. Delete the #include <iostream> line and re-enter it. You can see the auto-select boxes appearing.

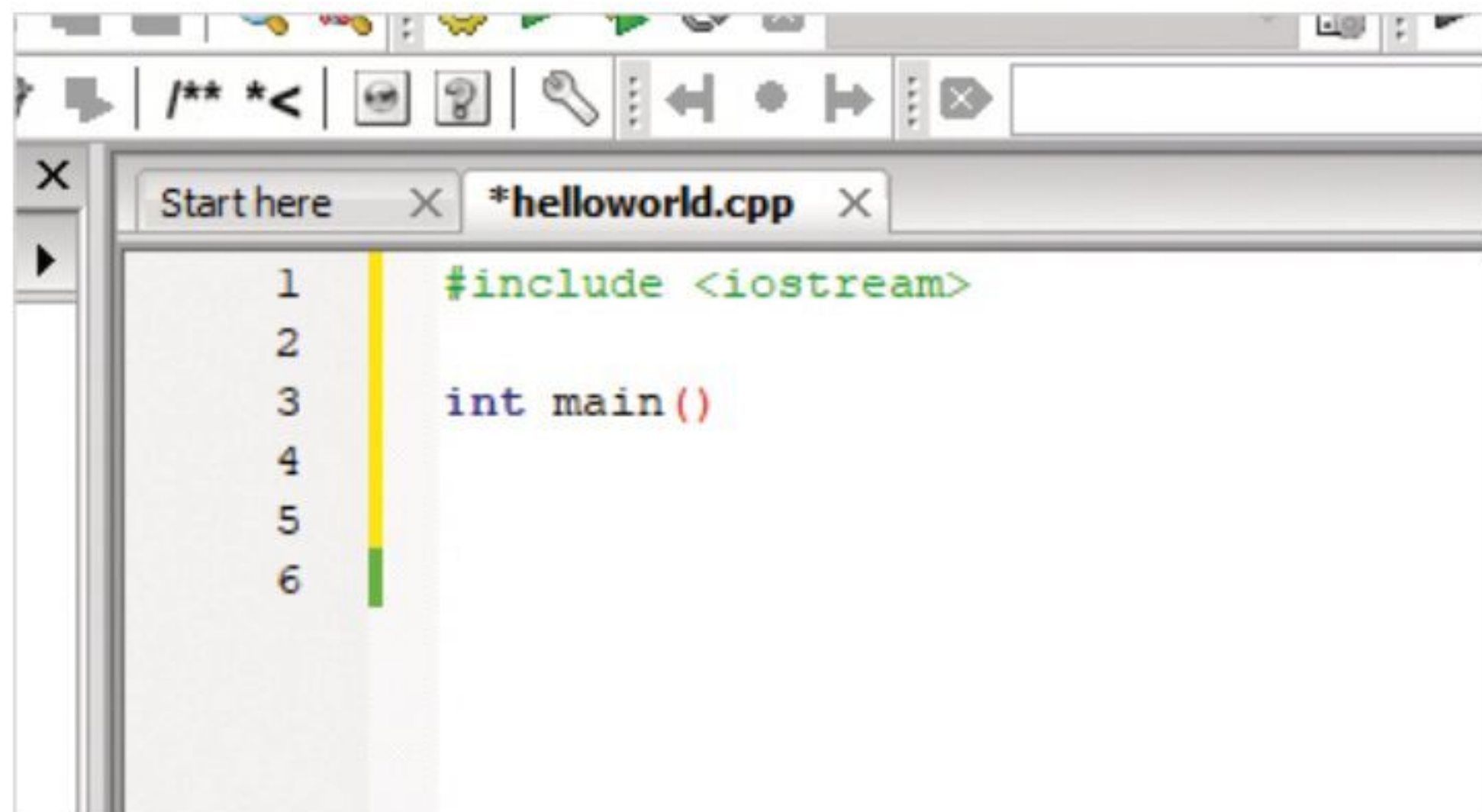


STEP 5

Auto-selection of commands is extremely handy and cuts out potential mistyping. Press Return to get to line 3, then enter:

```
int main()
```

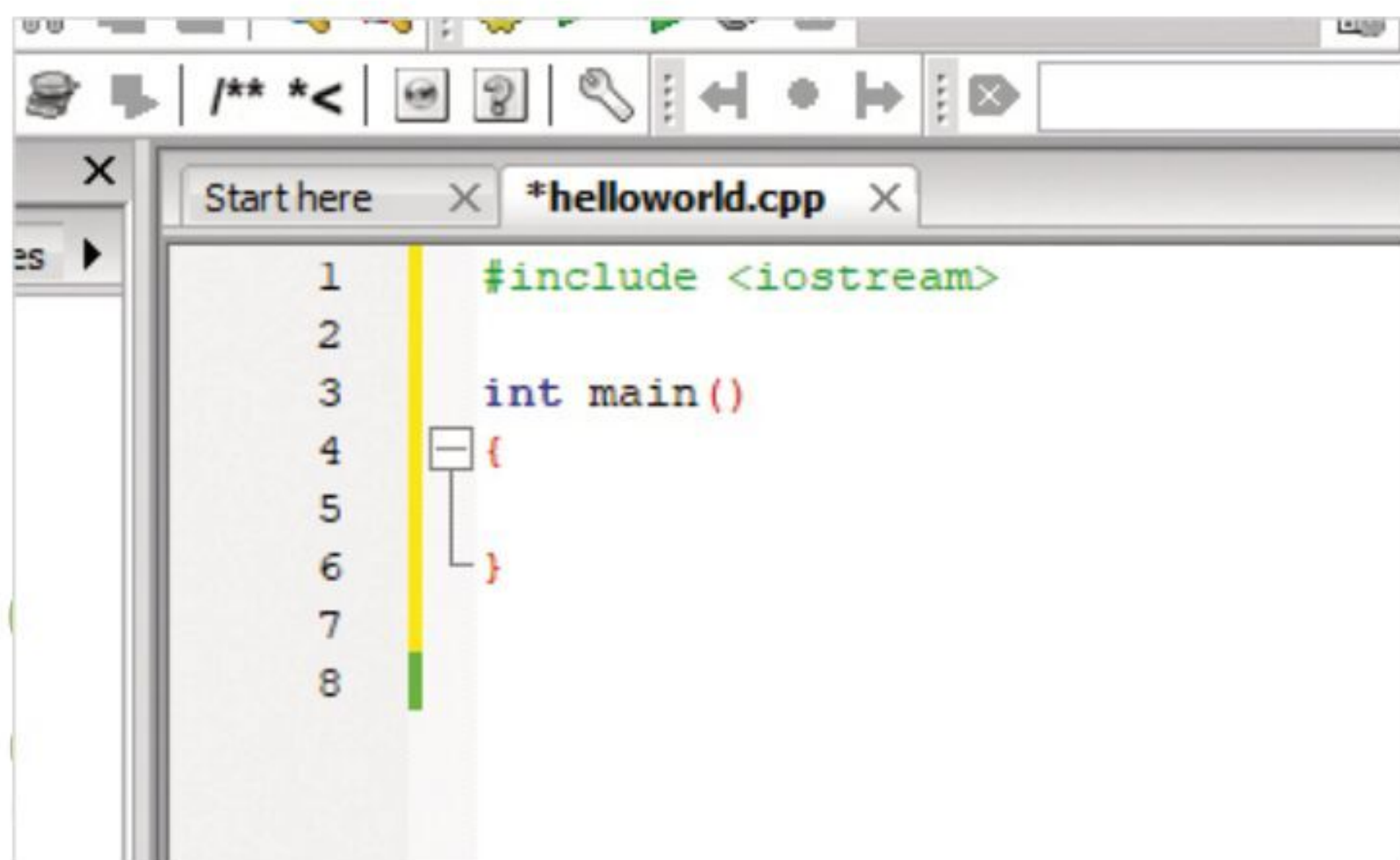
Note: there's no space between the brackets.

**STEP 6**

On the next line below `int main()`, enter a curly bracket:

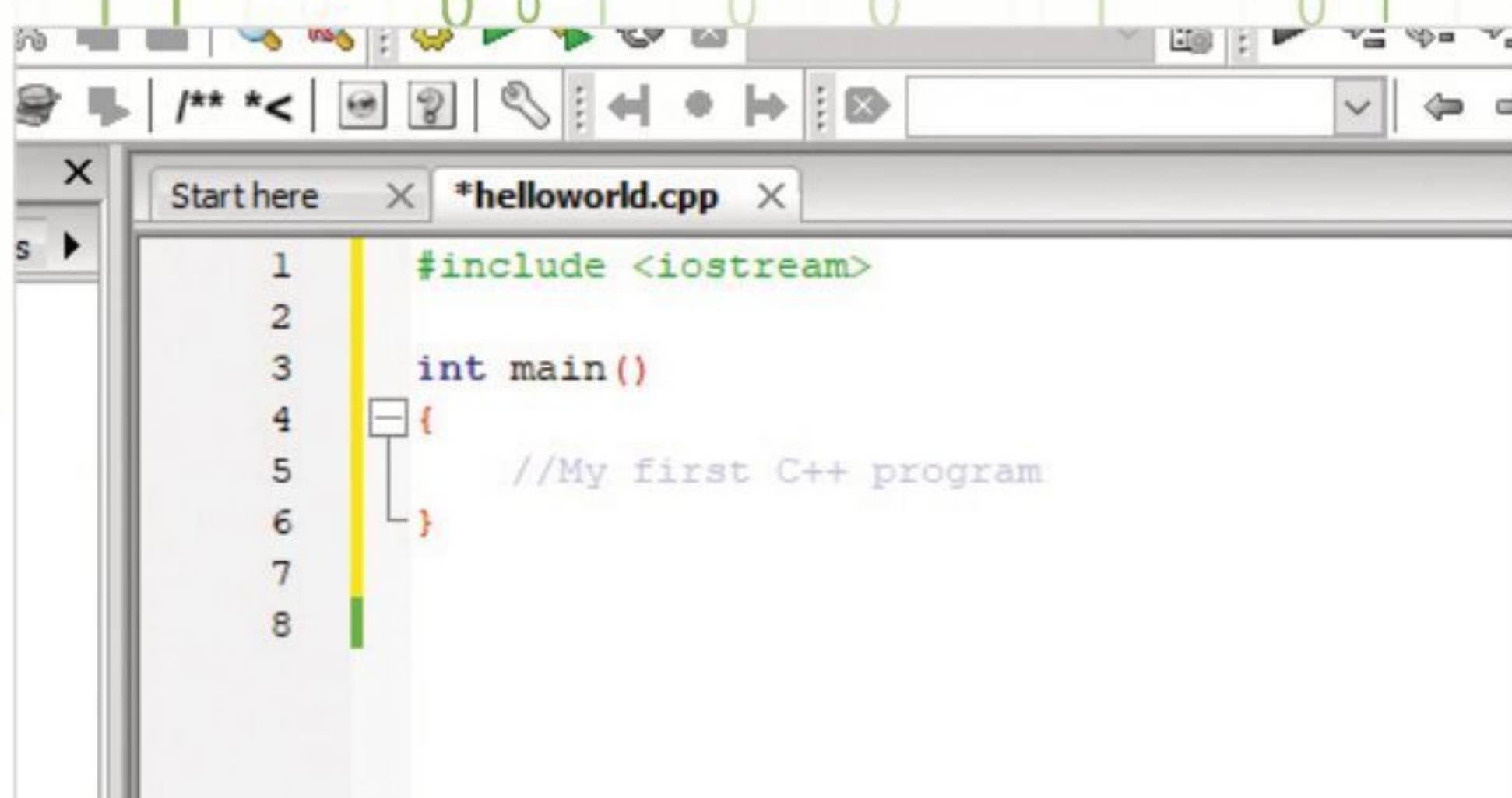
```
{
```

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.

**STEP 7**

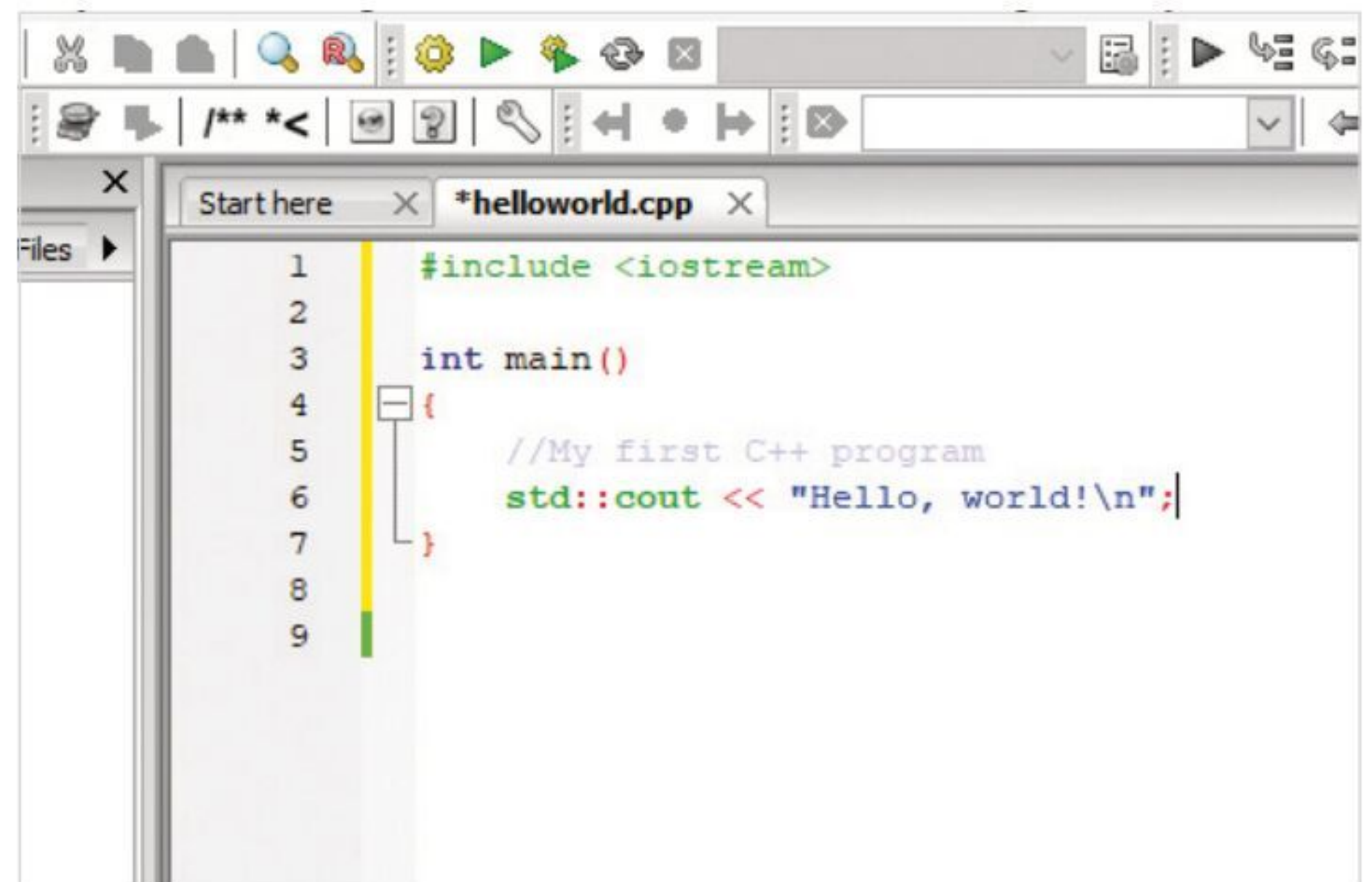
Notice that Code::Blocks has automatically created a corresponding closing curly bracket a couple of lines below, linking the pair, as well as a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

```
//My first C++ program
```

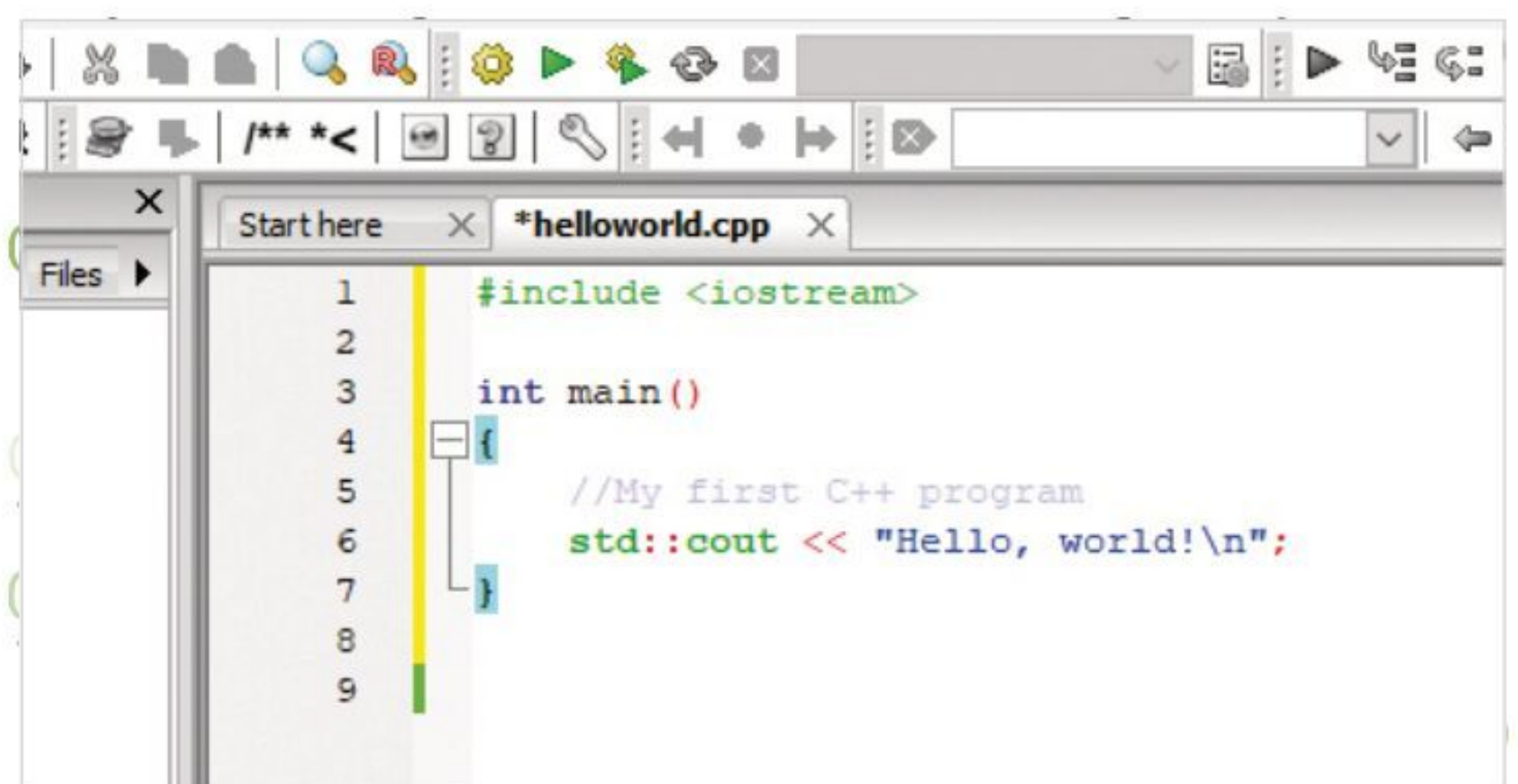
**STEP 8**

Note again the colour coding change. Press Return at the end of the previous step's line, and then enter:

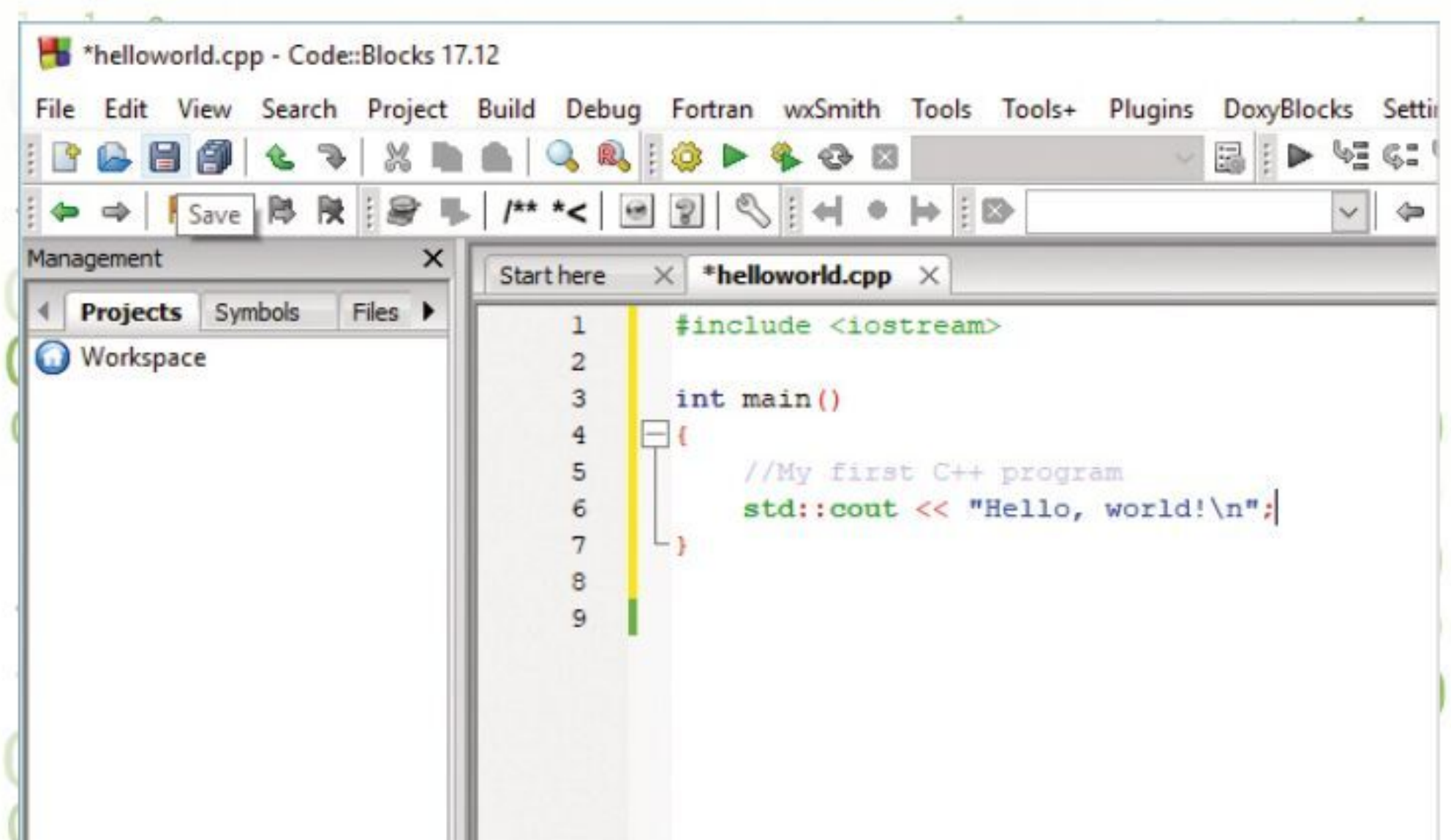
```
std::cout << "Hello, world!\n";
```

**STEP 9**

Just as before, Code::Blocks auto-completes the code you're entering, including placing a closing speech mark as soon as you enter the first. Don't forget the semicolon at the end of the line; this is one of the most important elements to a C++ program and we'll tell you why in the next section. For now, move the cursor down to the closing curly bracket and press Return.

**STEP 10**

That's all you need to do for the moment. It may not look terribly amazing but C++ is best absorbed in small chunks. Don't execute the code at the moment as you need to look at how a C++ program is structured first; then you can build and run the code. For now, click on Save, the single floppy disc icon.





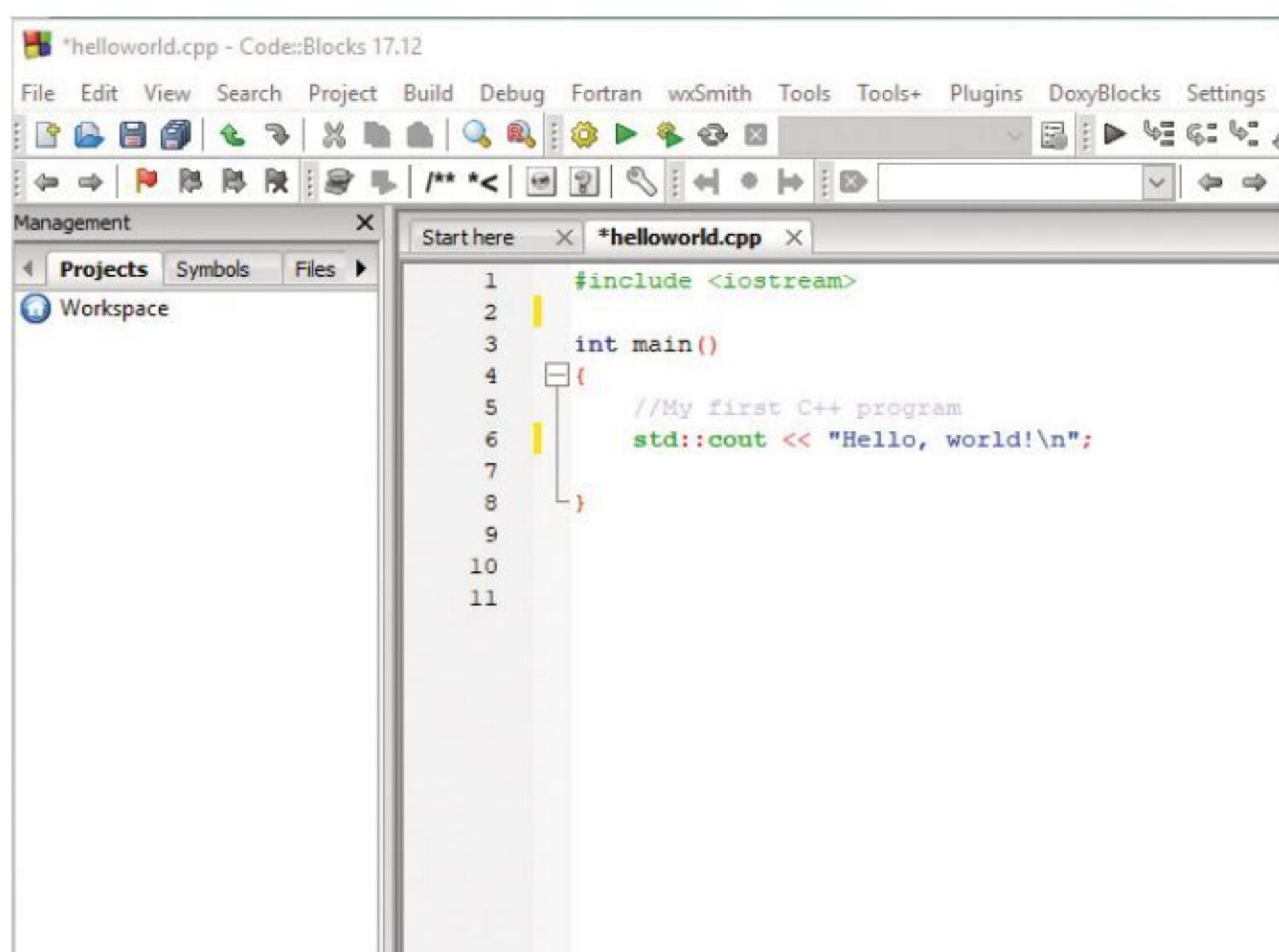
Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's actually get things moving and compile and execute, or run if you prefer, the program and see how it looks.

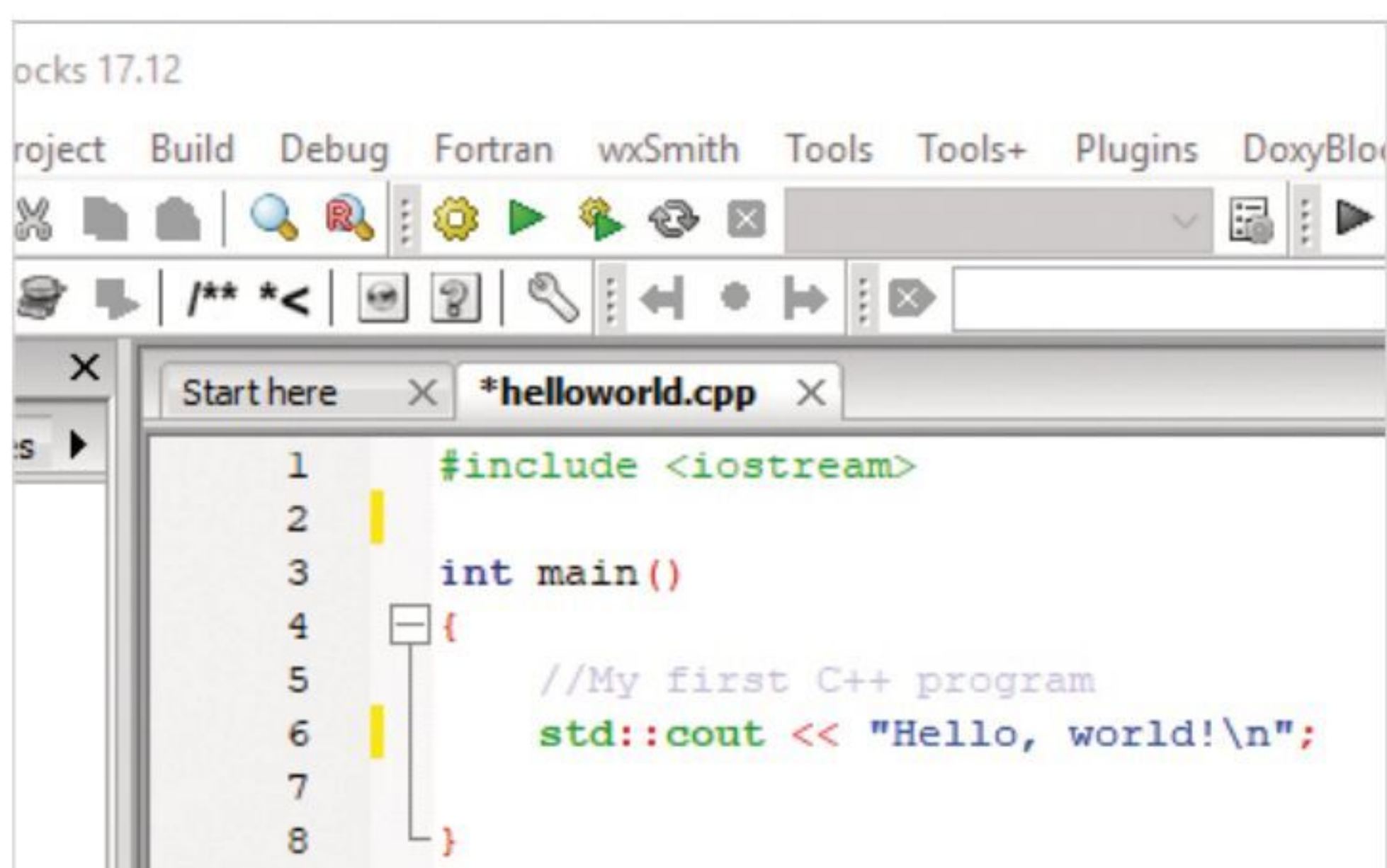
GREETINGS FROM C++

Compiling and executing C++ code from Code::Blocks is extraordinarily easy; it's just a matter of clicking an icon and seeing the result. Here's how it's done.

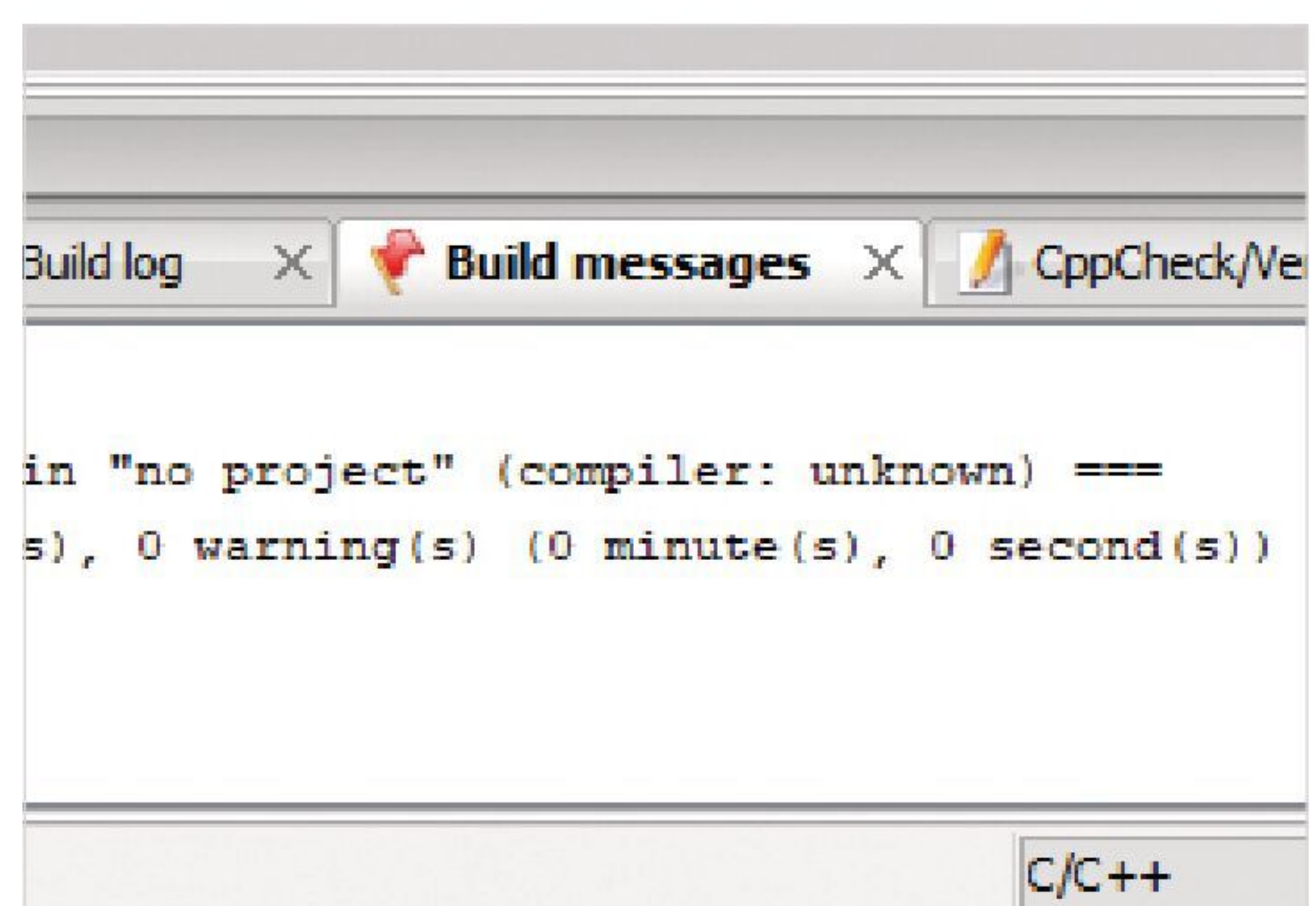
STEP 1 Open Code::Blocks, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as missing semicolons at the end of the `std::cout` line.



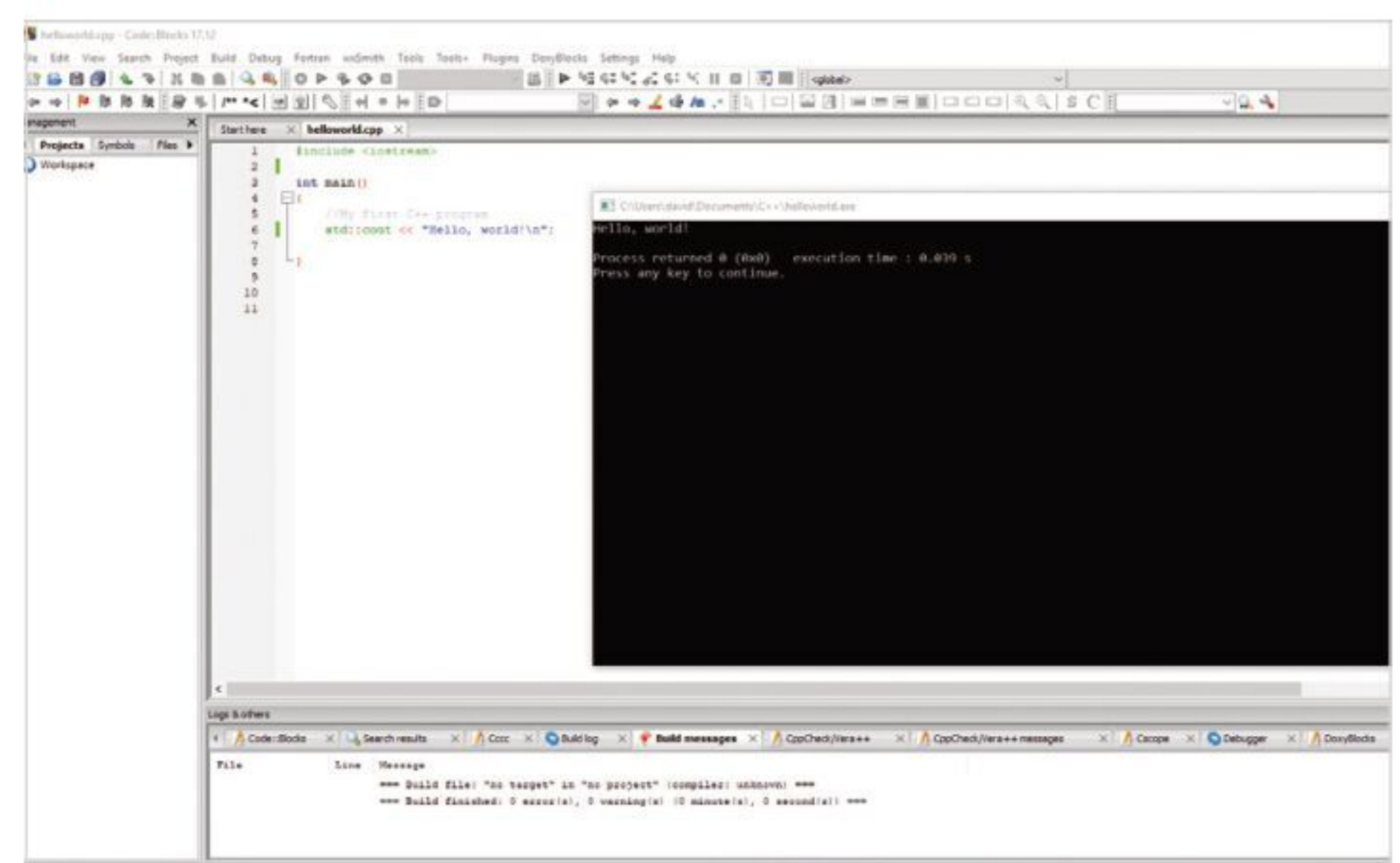
STEP 2 If your code is looking similar to the one in our screenshot, then look to the menu bar along the top of the screen. Under the Fortran entry in the topmost menu you can see a group of icons: a yellow cog, green play button and a cog/play button together. These are Build, Run, Build and Run functions.



STEP 3 Start by clicking on the Build icon, the yellow cog. At this point, your code has now been run through the Code::Blocks compiler and checked for any errors. You can see the results of the Build by looking to the bottom window pane. Any messages regarding the quality of the code are displayed here.



STEP 4 Now click on the Run icon, the green play button. A command line box appears on your screen displaying the words: Hello, world!, followed by the time it's taken to execute the code, and asking you press a key to continue. Well done, you just compiled and executed your first C++ program.





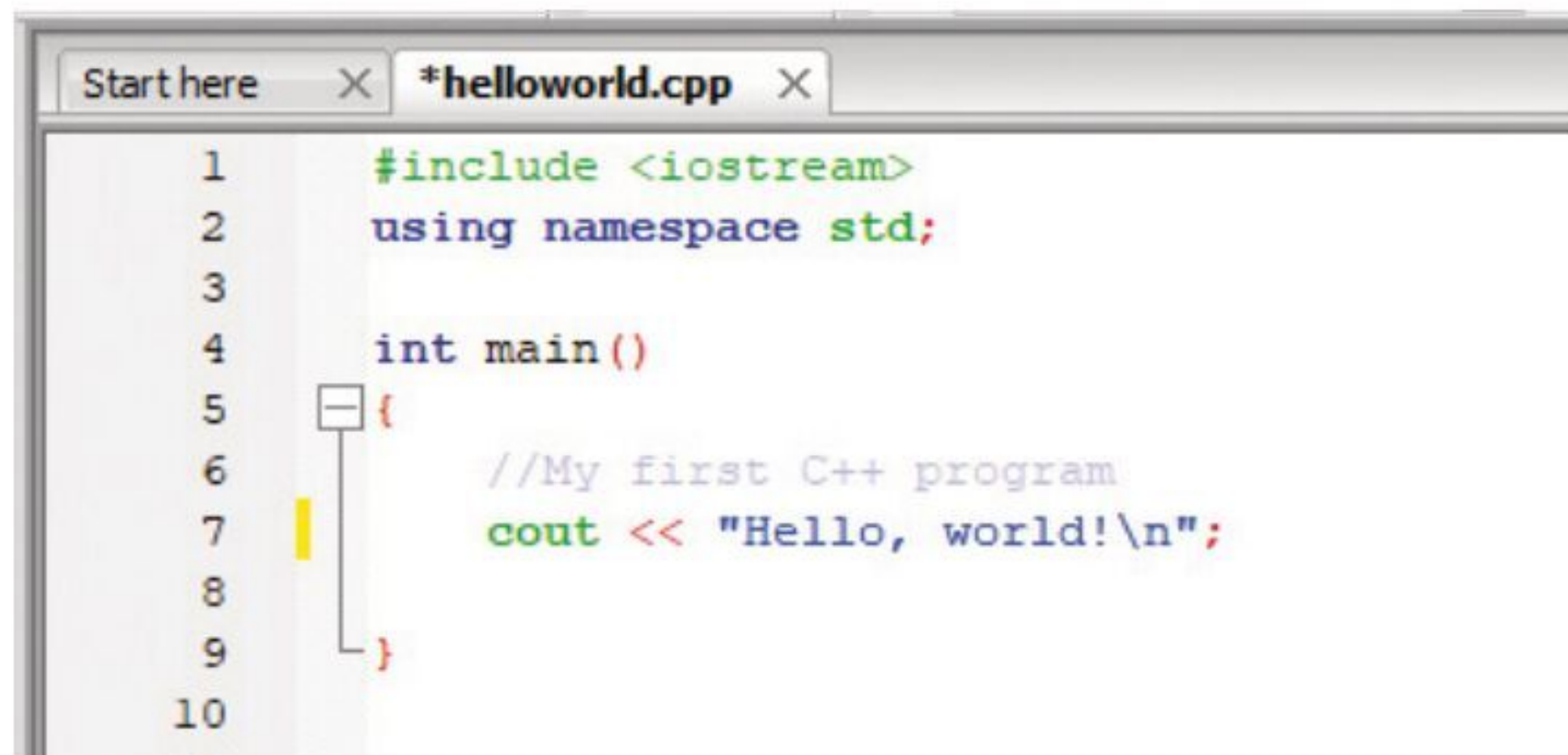
STEP 5

Pressing any key in the command line box closes it, returning you to Code::Blocks. Let's alter the code slightly. Under the #include line, enter:

```
using namespace std;
```

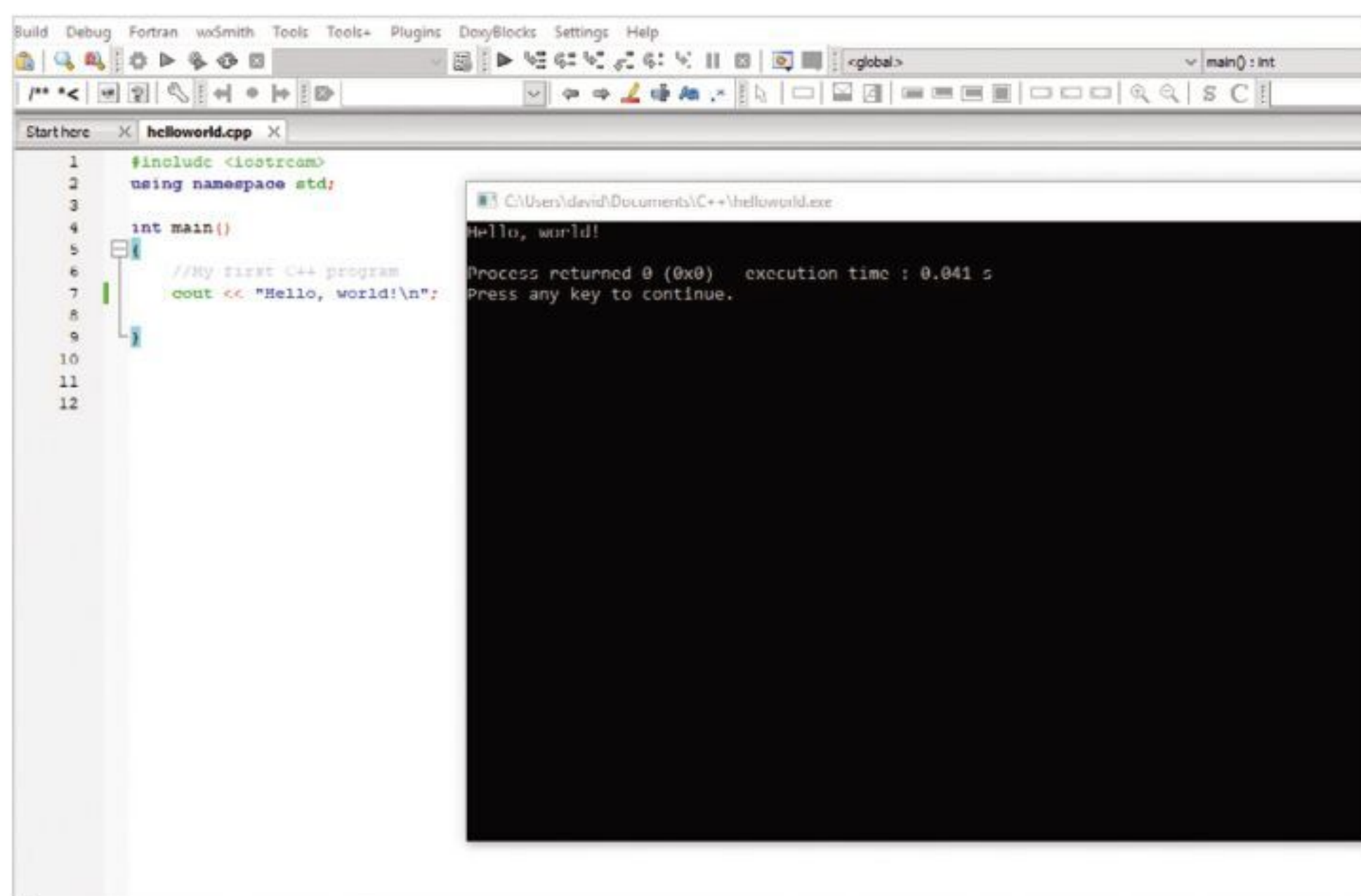
Then, delete the std:: part of the Cout line; like so:

```
cout << "Hello, world\n";
```



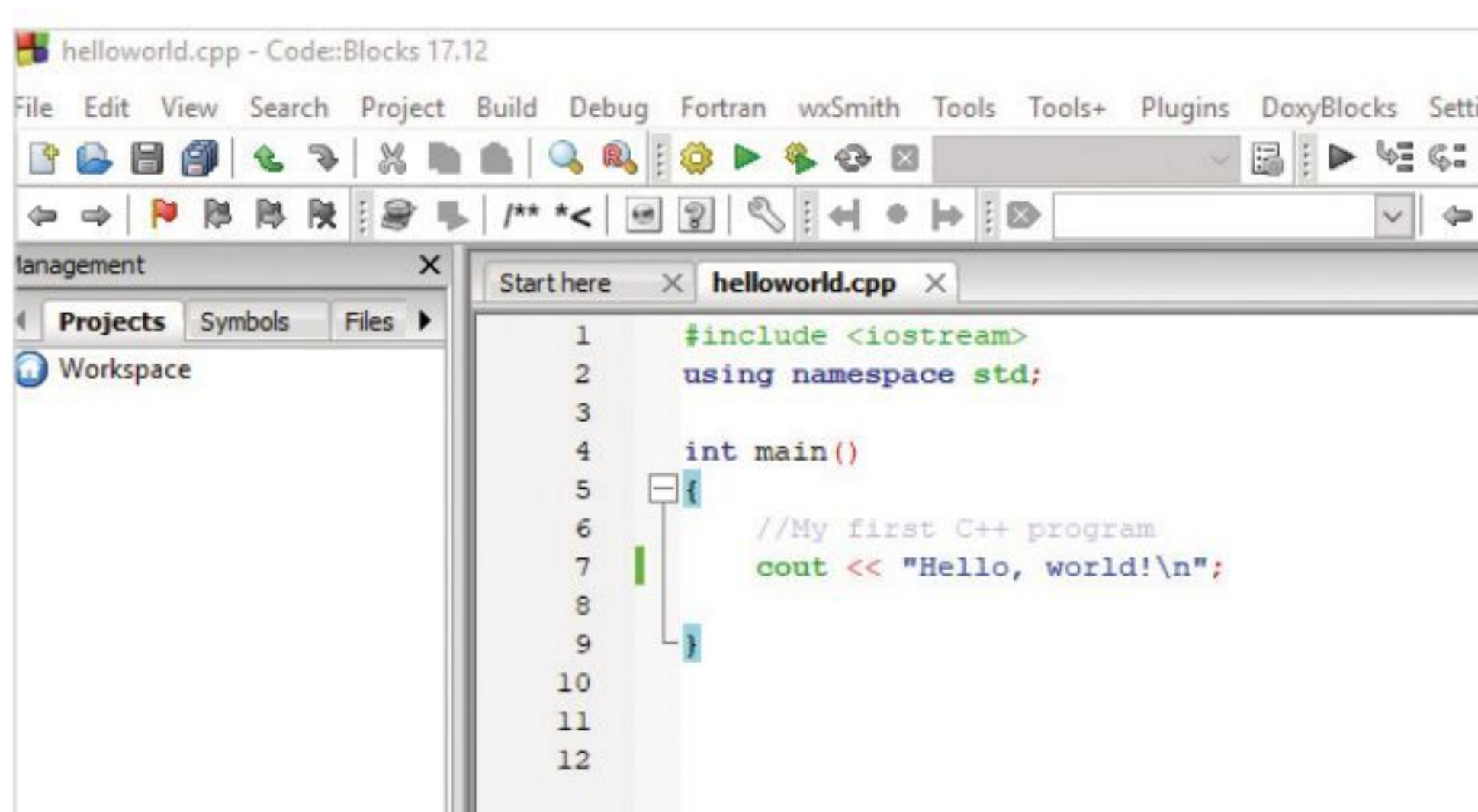
STEP 6

In order to apply the new changes to the code, you need to re-compile, build, and run it again. This time, however, you can simply click the Build/Run icon, the combined yellow cog and green play button.



STEP 7

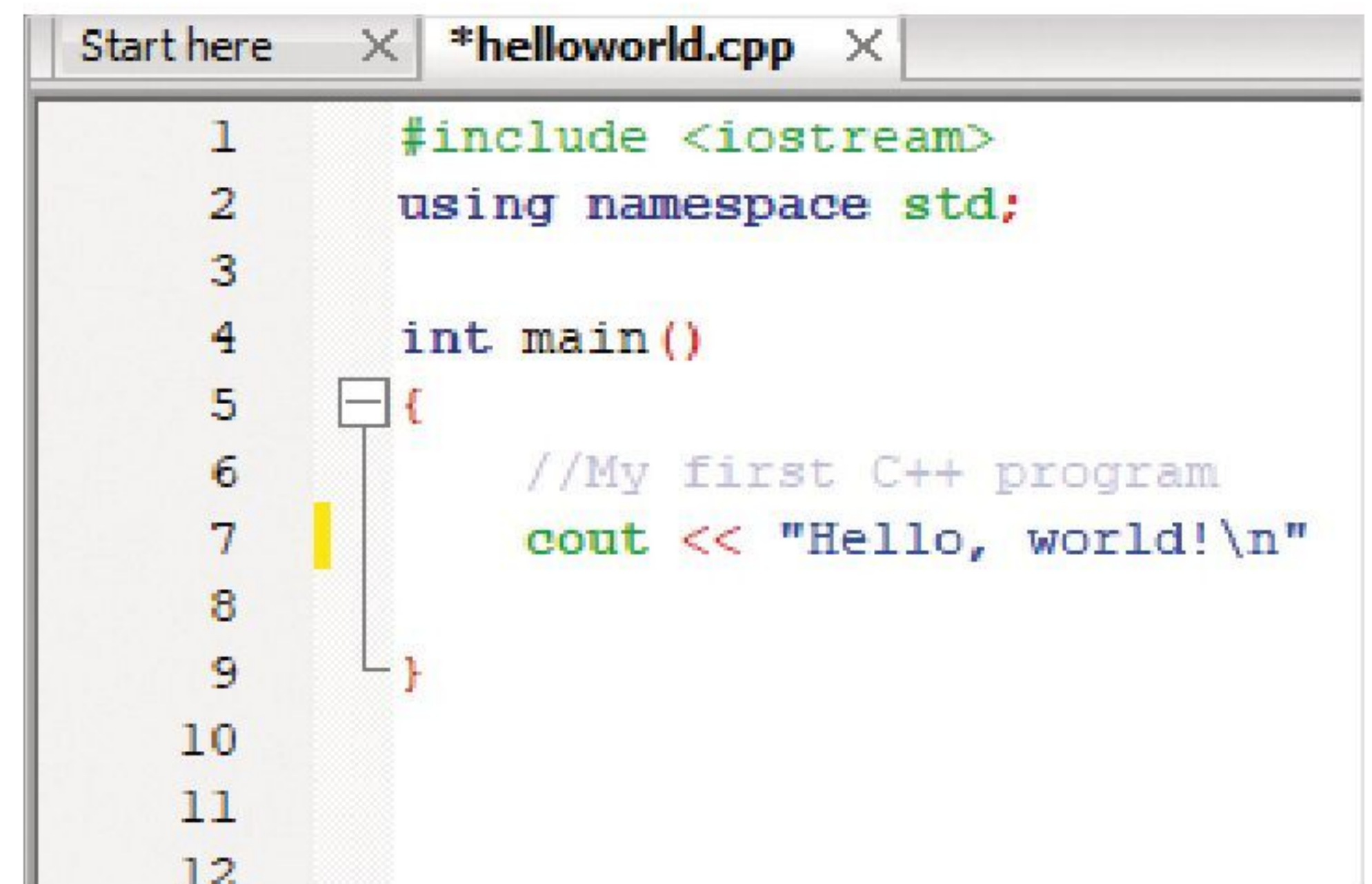
Just as we mentioned in the previous pages, you don't need to have std::cout if you already declare using namespace std; at the beginning of the code. We could have easily clicked the Build/Run icon to begin with but it's worth going through the available options. You can also see that by building and running, the file has been saved.



STEP 8

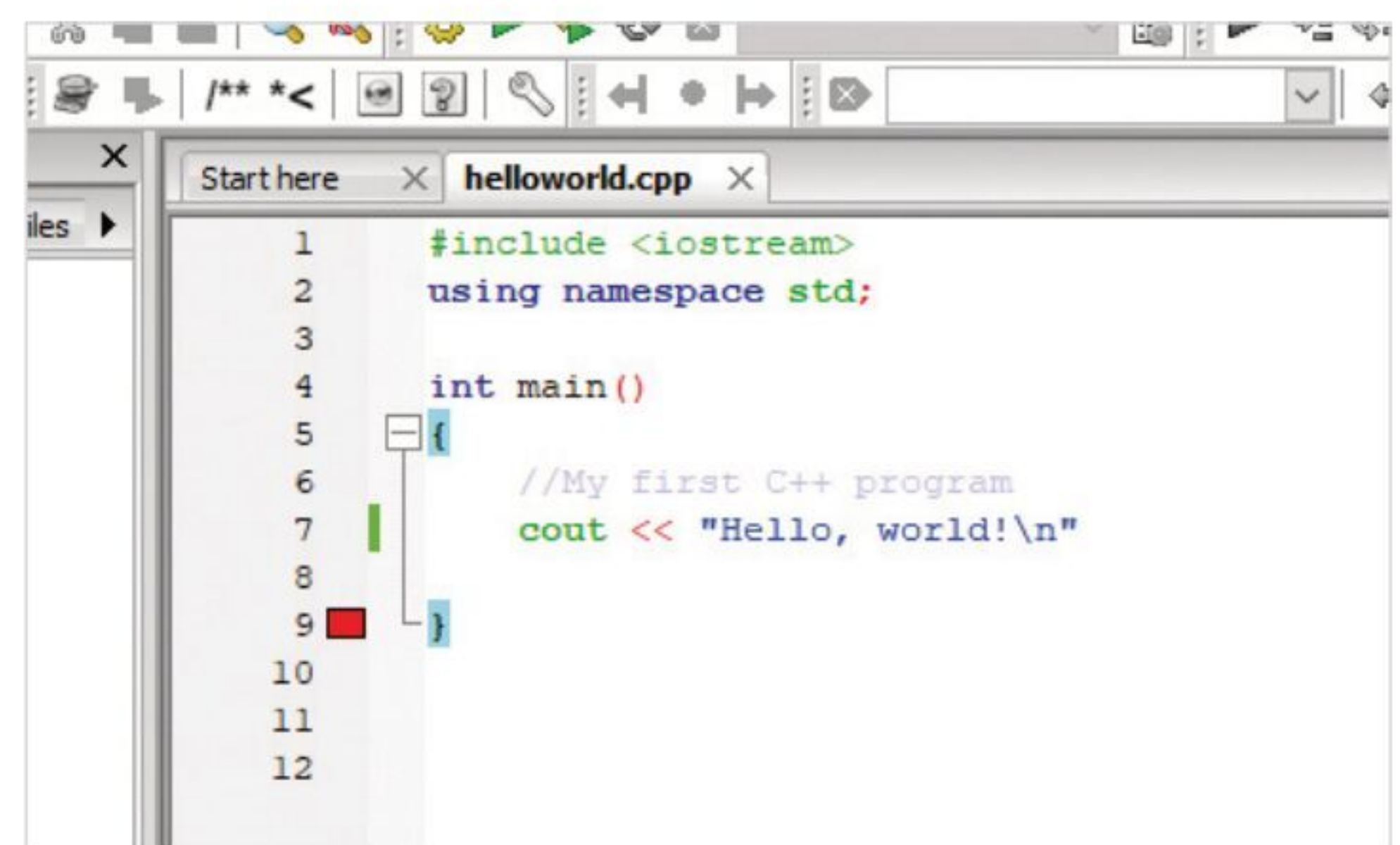
Create a deliberate error in the code. Remove the semicolon from the cout line, so it reads:

```
cout << "Hello, world!\n"
```



STEP 9

Now click the Build and Run icon again to apply the changes to the code. This time Code::Blocks refuses to execute the code, due to the error you put in. In the Log pane at the bottom of the screen you are informed of the error, in this case: Expected ';' before '}' token, indicating the missing semicolon.

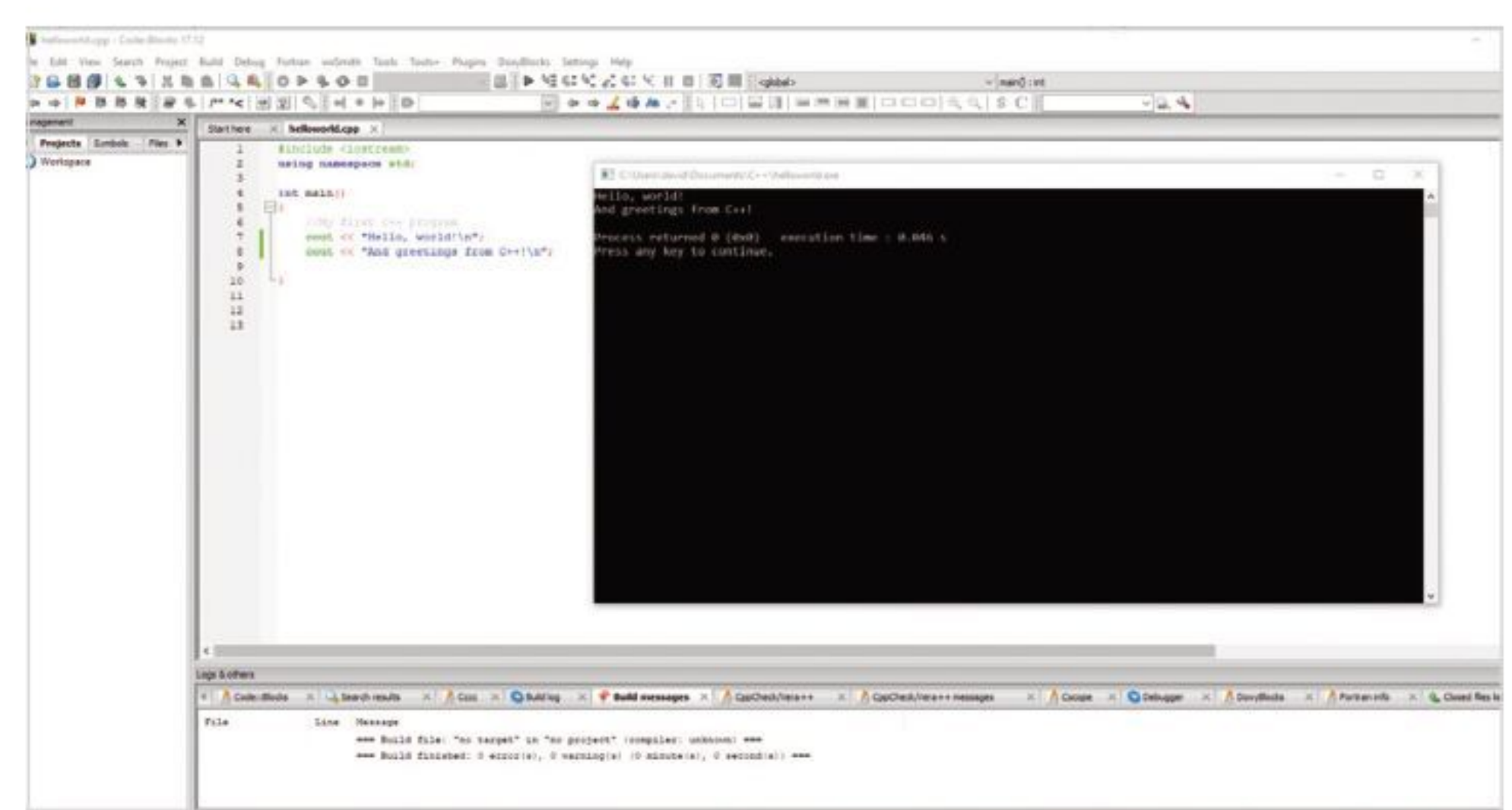


STEP 10

Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << "And greetings from C++!\n";
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.





Using Comments

While comments may seem like a minor element to the many lines of code that combine to make a game, application or even an entire operating system, in actual fact they're probably one of the most important factors.

THE IMPORTANCE OF COMMENTING

Comments inside code are basically human readable descriptions that detail what the code is doing at that particular point. They don't sound especially important but code without comments is one of the many frustrating areas of programming, regardless of whether you're a professional or just starting out.

In short, all code should be commented in such a manner as to effectively describe the purpose of a line, section, or individual elements. You should get in to the habit of commenting as much as possible, by imagining that someone who doesn't know anything about programming can pick up your code and understand what it's going to do simply by reading your comments.

In a professional environment, comments are vital to the success of the code and ultimately, the company. In an organisation, many programmers work in teams alongside engineers, other developers, hardware analysts and so on. If you're a part of the team that's writing a bespoke piece of software for the company, then your comments help save a lot of time should something go wrong, and another team member has to pick up and follow the trail to pinpoint the issue.

Place yourself in the shoes of someone whose job it is to find out what's wrong with a program. The program has in excess of 800,000 lines of code, spread across several different modules. You can soon appreciate the need for a little help from the original programmers in the form of a good comment.

The best comments are always concise and link the code logically, detailing what happens when the program hits this line or section. You don't need to comment on every line. Something along the lines of: if $x==0$ doesn't require you to comment that if x equals zero then do something; that's going to be obvious to the reader. However, if x

equalling zero is something that drastically changes the program for the user, such as, they've run out of lives, then it certainly needs to be commented on.

Even if the code is your own, you should write comments as if you were going to publicly share it with others. This way you can return to that code and always understand what it was you did or where it was you went wrong or what worked brilliantly.

Comments are good practise and once you understand how to add a comment where needed, you soon do it as if it's second nature.

```
DEFB 26h,30h,32h,26h,30h,32h,0,0,32h,72h,73h,32h,72h,73h,32h
DEFB 60h,61h,32h,4Ch,4Dh,32h,4Ch,99h,32h,4Ch,4Dh,32h,4Ch,4Dh
DEFB 32h,4Ch,99h,32h,5Bh,5Ch,32h,56h,57h,32h,33h,0CDh,32h,33h
DEFB 34h,32h,33h,34h,32h,33h,0CDh,32h,40h,41h,32h,66h,67h,64h
DEFB 66h,67h,32h,72h,73h,64h,4Ch,4Dh,32h,56h,57h,32h,80h,0CBh
DEFB 19h,80h,0,19h,80h,81h,32h,80h,0CBh,0FFh

T858C:
DEFB 80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h,56h,66h
DEFB 56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h
DEFB 56h,56h,56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h
DEFB 51h,51h,56h,66h,56h,56h,80h,72h,66h,60h,56h,66h,56h,40h
DEFB 56h,66h,80h,66h,56h,56h,56h,56h

;
; Game restart point
;
START: XOR A
LD (SHEET),A
LD (KEMP),A
LD (DEMO),A
LD (B845B),A
LD (B8458),A
LD A,2 ;Initial lives count
LD (NOMEN),A
LD HL,T845C
SET 0,(HL)
LD HL,SCREEN
LD DE,SCREEN+1
LD BC,17FFh ;Clear screen image
LD (HL),0
LDIR
LD HL,0A000h ;Title screen bitmap
LD DE,SCREEN
LD BC,4096
LDIR
LD HL,SCREEN + 800h + 1*32 + 29
LD DE,MANDAT+64
LD C,0
CALL DRWFIX
LD HL,0FC00h ;Attributes for the last room
LD DE,ATTR ;(top third)
LD BC,256
LDIR
LD HL,09E00h ;Attributes for title screen
LD BC,512 ;(bottom two-thirds)
LDIR
LD BC,31
DI
XOR A
R8621: IN E,(C)
OR E
DJNZ R8621 ;$-03
AND 20h
JR NZ,R862F ;$+07
LD A,1
LD (KEMP),A
R862F: LD IY,T846E
CALL C92DC
JP NZ,L8684
XOR A
LD (EUGHGT),A
```





C++ COMMENTS

Commenting in C++ involves using a double forward slash '/', or a forward slash and an asterisk, '/*'. You've already seen some brief examples but this is how they work.

STEP 1

Using the Hello World code as an example, you can easily comment on different sections of the code using the double forward slash:

```
//My first C++ program
cout << "Hello, world!\n";
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8
9
10 }
11
12
13
```

STEP 2

However, you can also add comments to the end of a line of code, to describe in a better way what's going on:

```
cout << "Hello, world!\n"; //This line outputs the
words 'Hello, world!'. The \n denotes a new line.
```

Note, you don't have to put a semicolon at the end of a comment. This is because it's a line in the code that's ignored by the compiler.

```
Start here x *helloworld.cpp x
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n"; //This line outputs the words 'Hello, world!'. The \n
8
9
10 }
11
12
13
14
```

STEP 3

You can comment out several lines by using the forward slash and asterisk:

```
/* This comment can
   cover several lines
   without the need to add more slashes */
```

Just remember to finish the block comment with the opposite asterisk and forward slash.

```
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8      cout << "And greetings from C++";
9
10     /* This comment can
11        cover several lines
12        without the need to add more slashes */
13
14
15 }
```

STEP 4

Be careful when commenting, especially with block comments. It's very easy to forget to add the closing asterisk and forward slash and thus negate any code that falls inside the comment block.

```
Start here x *helloworld.cpp x
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8      cout << "And greetings from C++";
9
10     /* This comment can
11        cover several lines
12        without the need to add more slashes
13
14     cout << "This line is now being ignored by the compiler!";
15
16
17 }
18
19
20
```

STEP 5

Obviously if you try and build and execute the code it errors out, complaining of a missing curly bracket '}' to finish off the block of code. If you've made the error a few times, then it can be time consuming to go back and rectify. Thankfully, the colour coding in Code::Blocks helps identify comments from code.

```
Start here x helloworld.cpp x
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8      cout << "And greetings from C++";
9
10     /* This comment can
11        cover several lines
12        without the need to add more slashes
13
14     cout << "This line is now being ignored by the compiler!";
15
16
17 }
18
```

STEP 6

If you're using block comments, it's good practise in C++ to add an asterisk to each new line of the comment block. This also helps you to remember to close the comment block off before continuing with the code:

```
/* This comment can
 * cover several lines
 * without the need to add more slashes */
```

```
4  int main()
5  {
6      //My first C++ program
7      cout << "Hello, world!\n";
8      cout << "And greetings from C++\n";
9
10     /* This comment can
11        * cover several lines
12        * without the need to add more slashes */
13
14     cout << "This line is now being ignored by the compiler!\n";
15
16
17 }
18
```


Variables

Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that 'a' equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

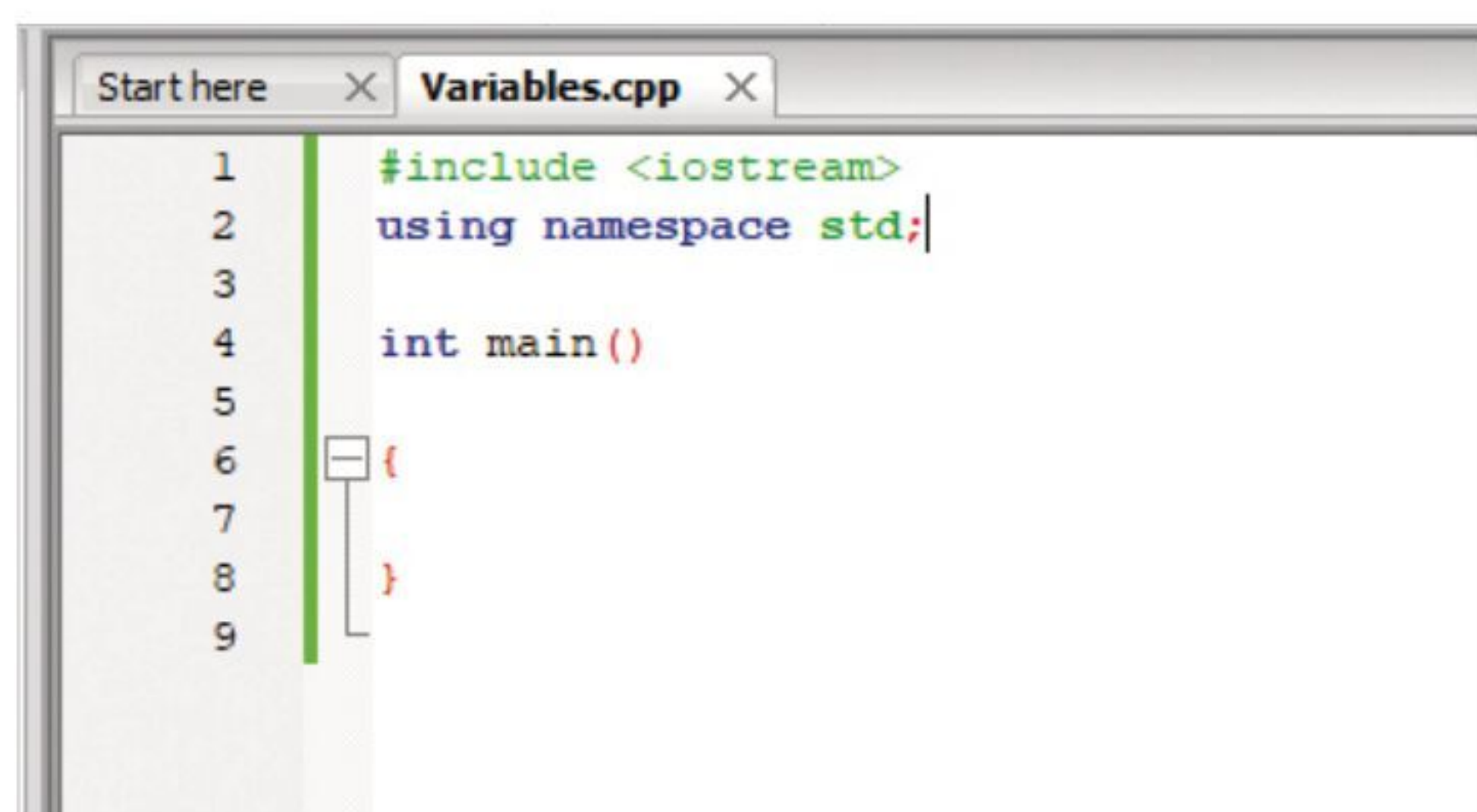
THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variables you can declare. Here's how it works.

STEP 1 Open up a new, blank C++ file and enter the usual code headers:

```
#include <iostream>
using namespace std;

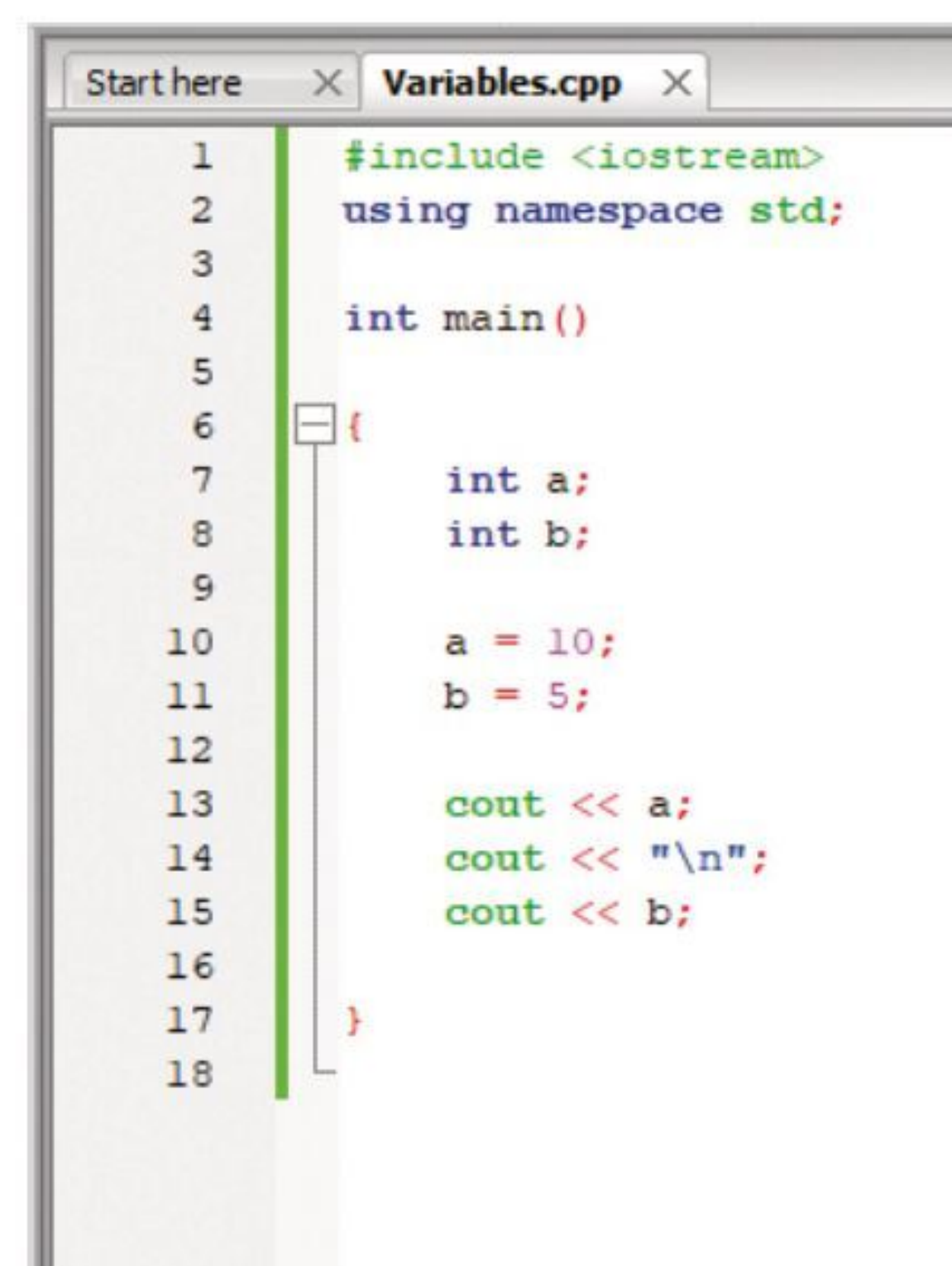
int main()
{
}
```



STEP 3 You can build and run the code but it won't do much, other than store the values 10 and 5 to the integers a and b. To output the contents of the variables, add:

```
cout << a;
cout << "\n";
cout << b;
```

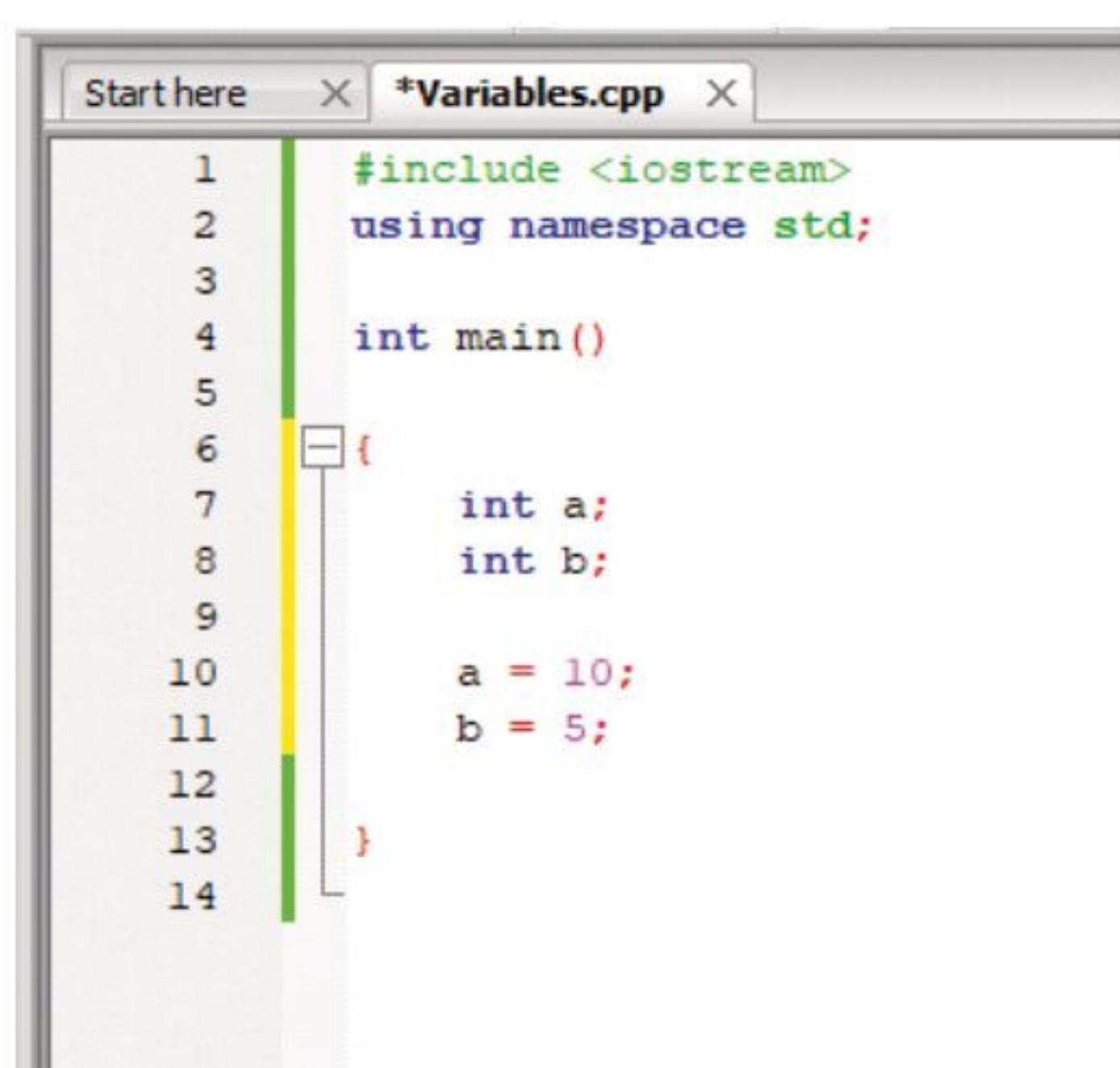
The cout << "\n"; part simply places a new line between the output of 10 and 5.



STEP 2 Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets, enter:

```
int a;
int b;

a = 10;
b = 5;
```

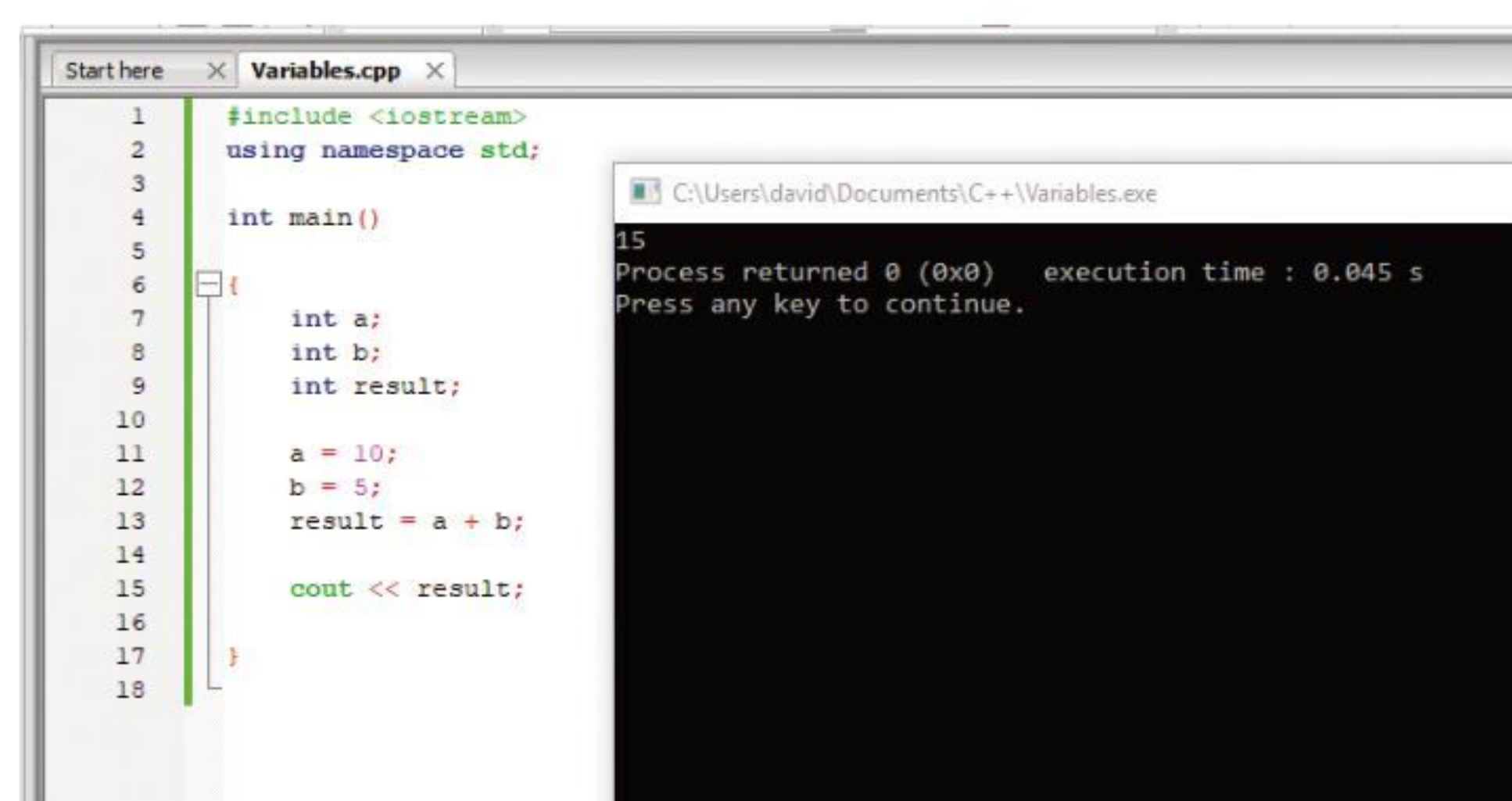


STEP 4 Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;

result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.

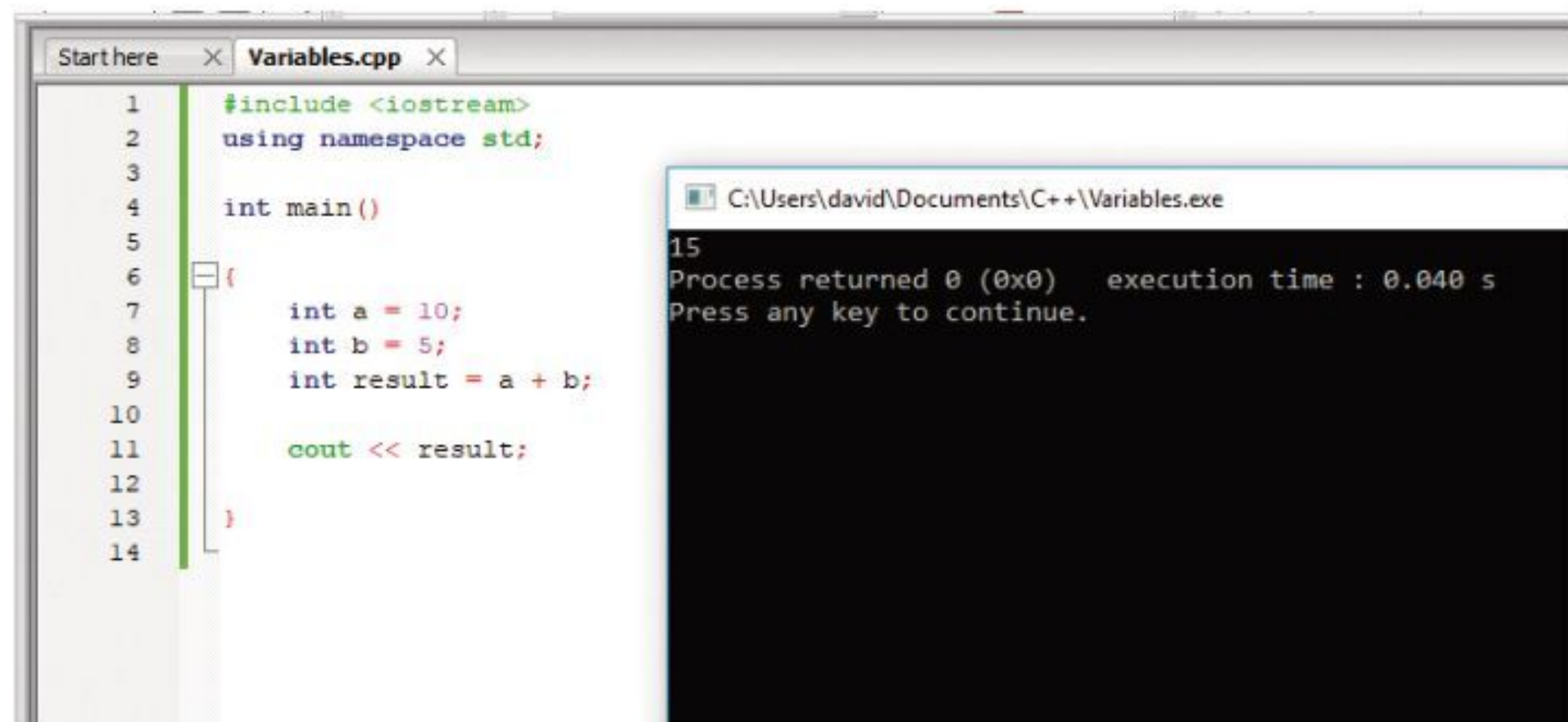


STEP 5

You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;

cout << result;
```

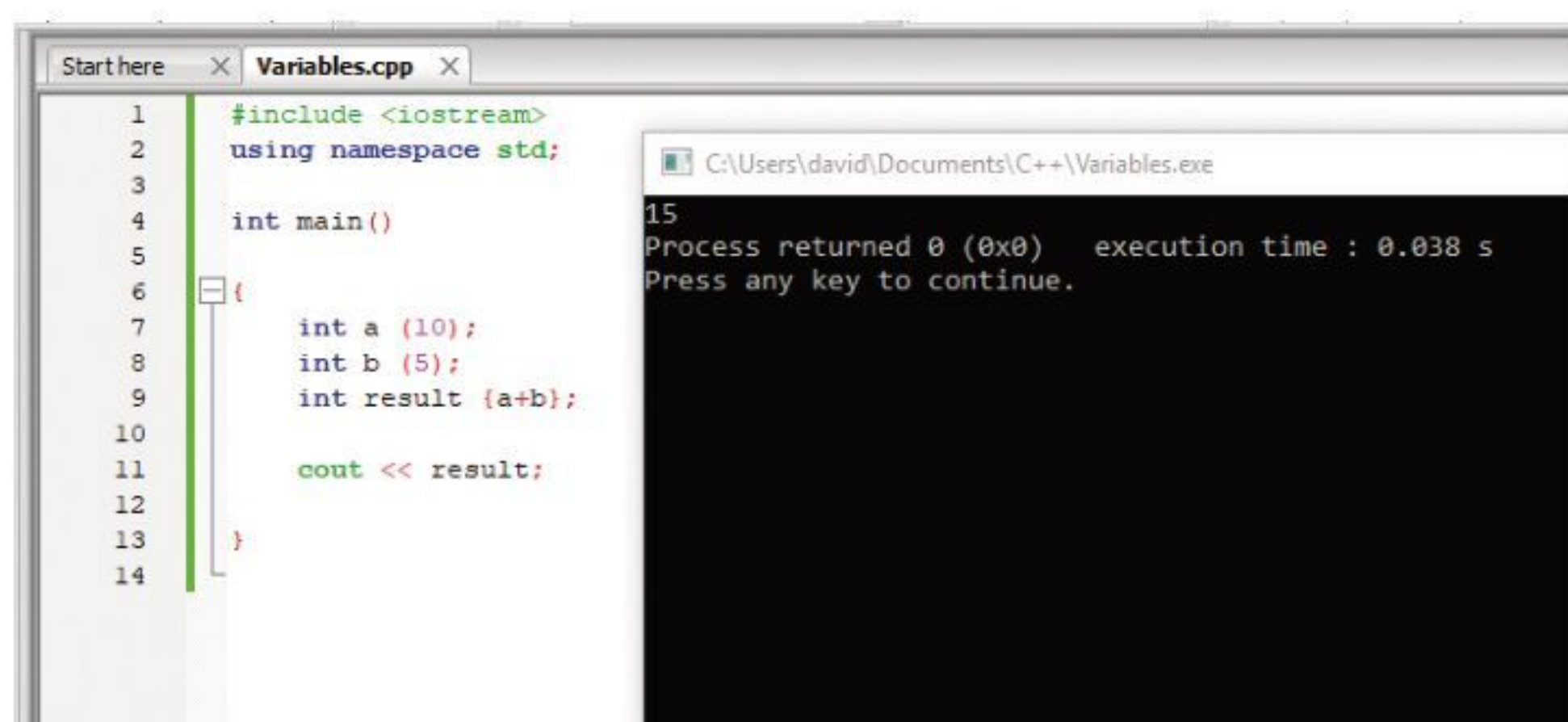
**STEP 6**

Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

Then, from the C++ 2011 standard, using curly brackets:

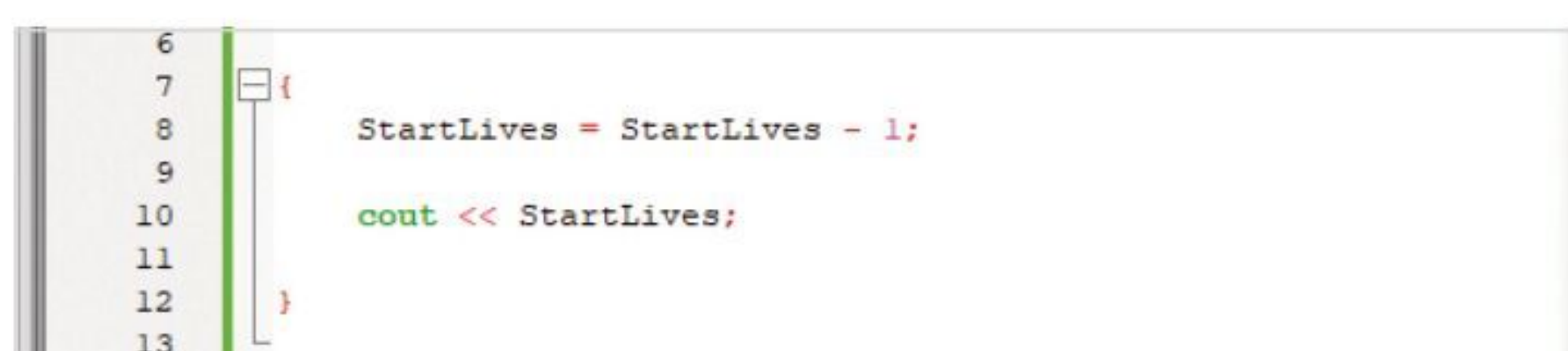
```
int result {a+b};
```

**STEP 7**

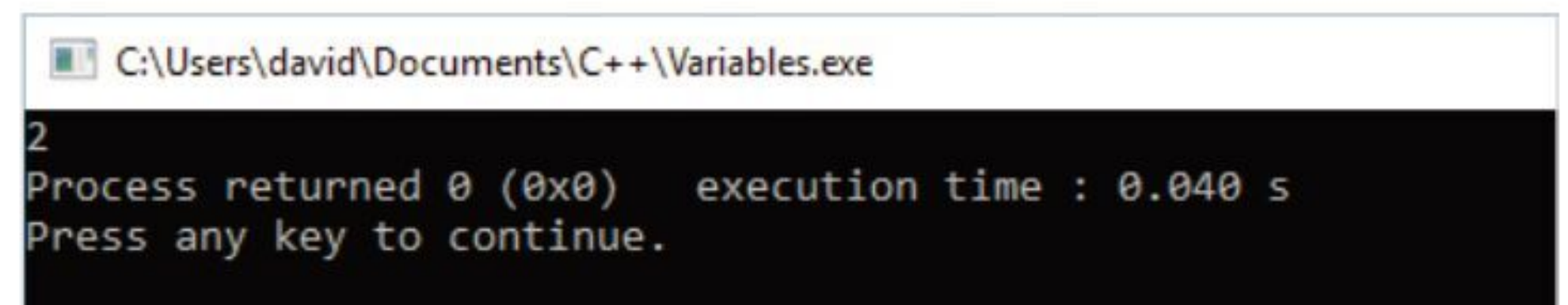
You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()
{
    startLives = StartLives - 1;
    cout << StartLives;
}
```

**STEP 8**

The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.

**STEP 9**

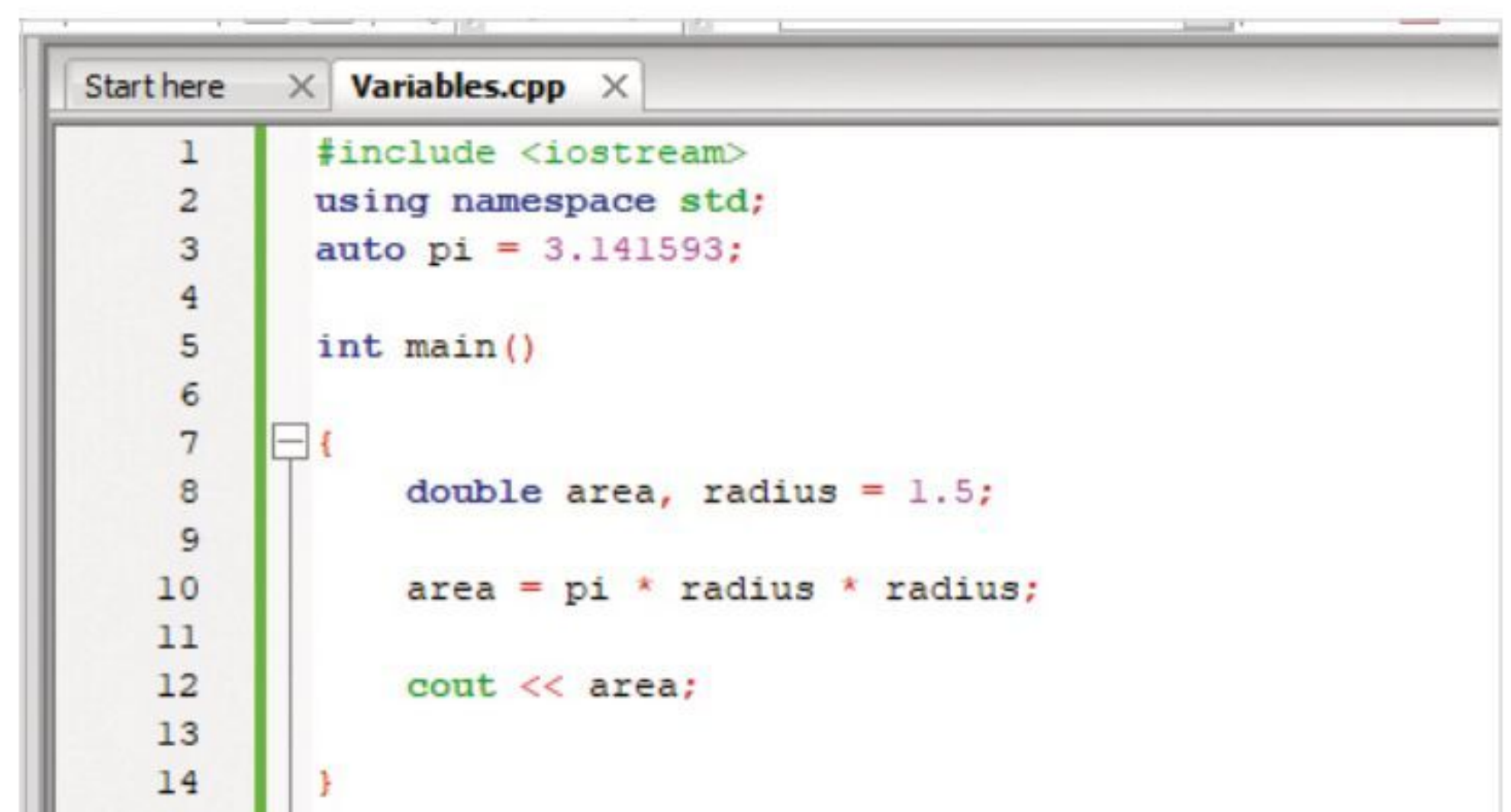
The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

```
#include <iostream>
using namespace std;
auto pi = 3.141593;

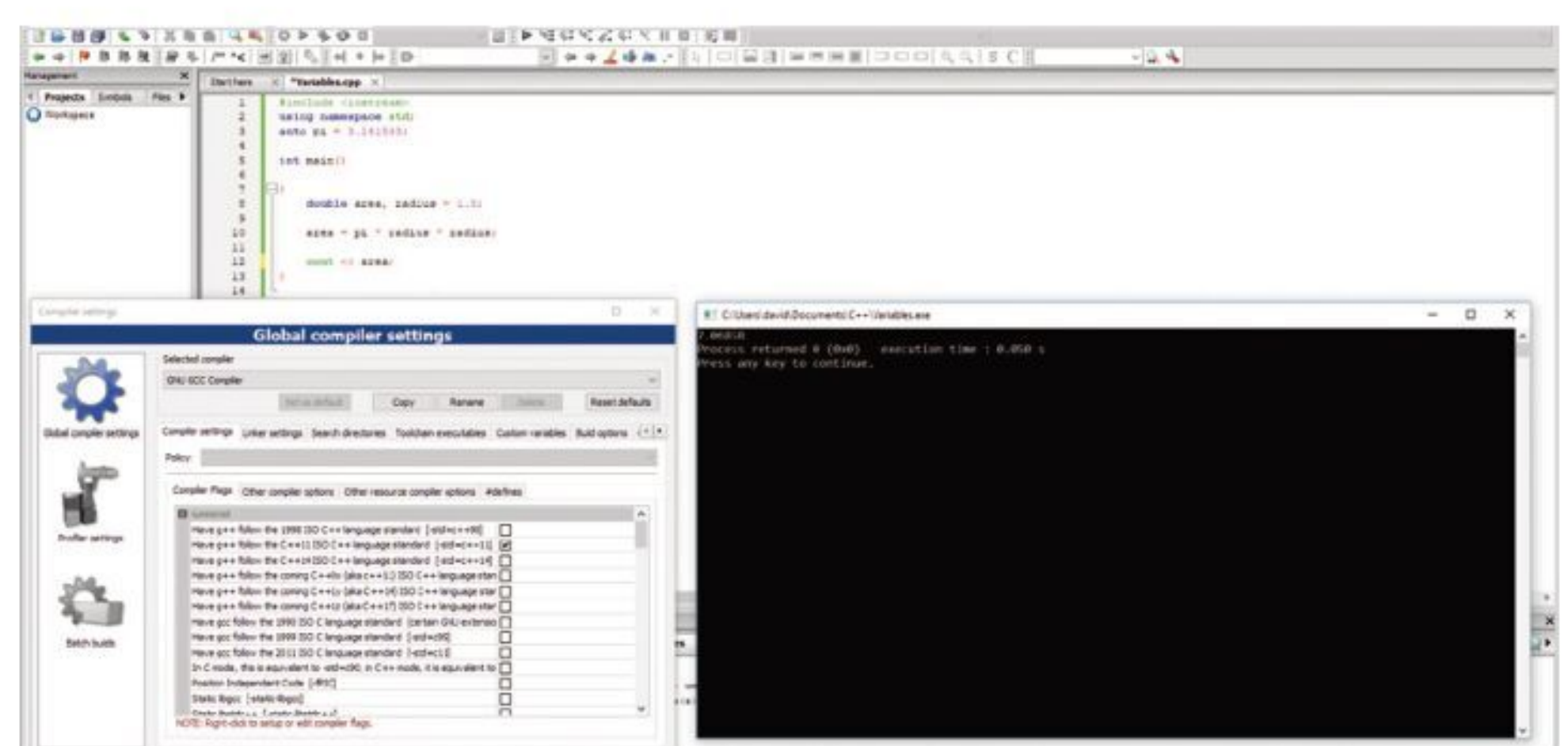
int main()
{
    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;
}
```

**STEP 10**

A couple of new elements here: first, auto won't work unless you go to Settings > Compiler and tick the box labelled 'Have G++ follow the C++11 ISO C++ Language Standard [-std=c++11]'. Then, the new data type, double, which means double-precision floating point value. Enable C++11, then build and run the code. The result should be 7.06858.





Data Types

Variables, as we've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

The seven basic data types are:

TYPE	COMMAND
Integer	<code>Integer</code>
Floating Point	<code>float</code>
Character	<code>char</code>
Boolean	<code>bool</code>
Double Floating Point	<code>double</code>
Wide Character	<code>wchar_t</code>
No Value	<code>void</code>

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295

Signed int = -2147483648 to 2147483647

Short int = -32768 to 32767

Unsigned Short int = 0 to 65,535

Signed Short int = -32768 to 32767

Long int = -2147483647 to 2147483647

Signed Long int = -2147483647 to 2147483647

Unsigned Long int = 0 to 4294967295

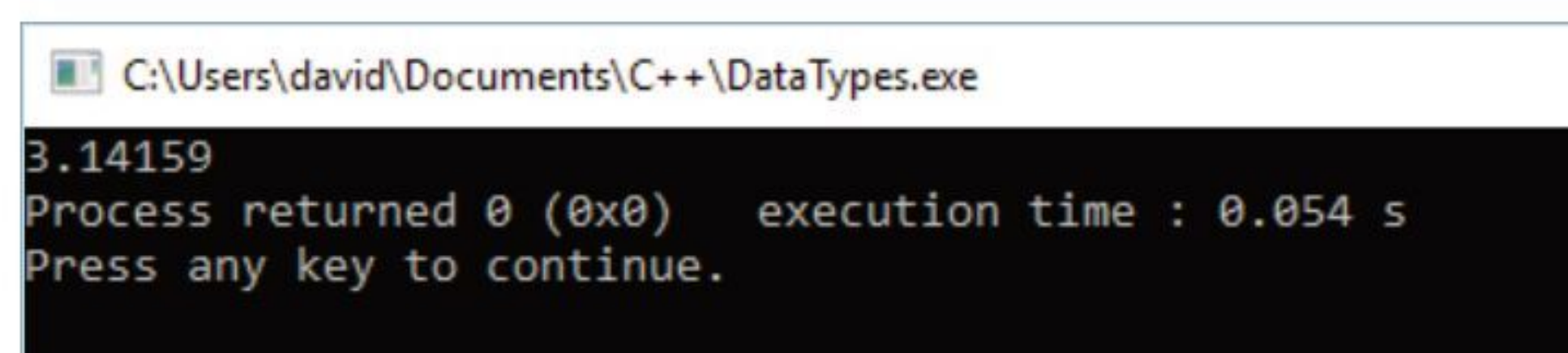
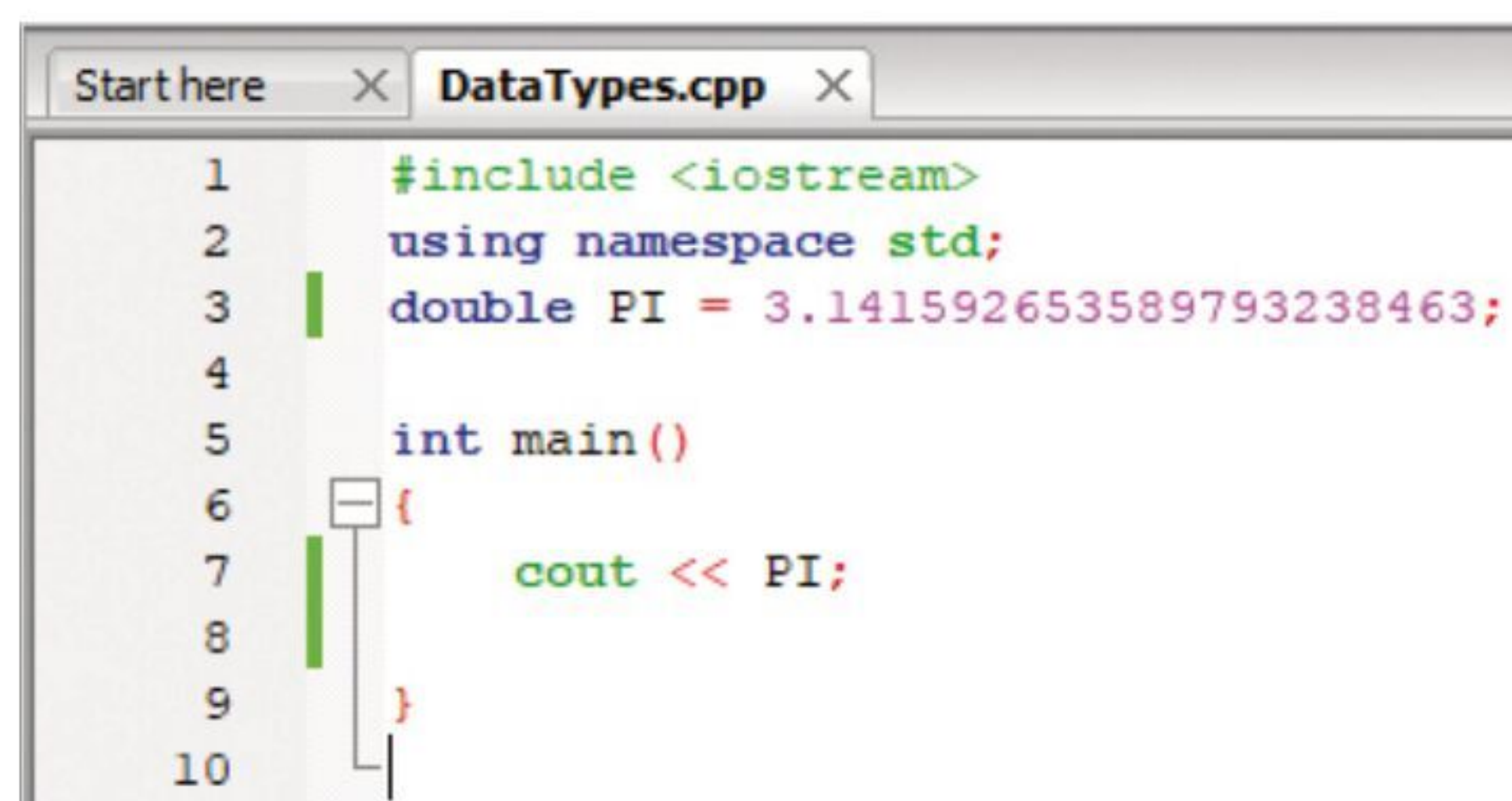
Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for

incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a `cout.precision()` function and adding a value inside the brackets, but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;
}
```



Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

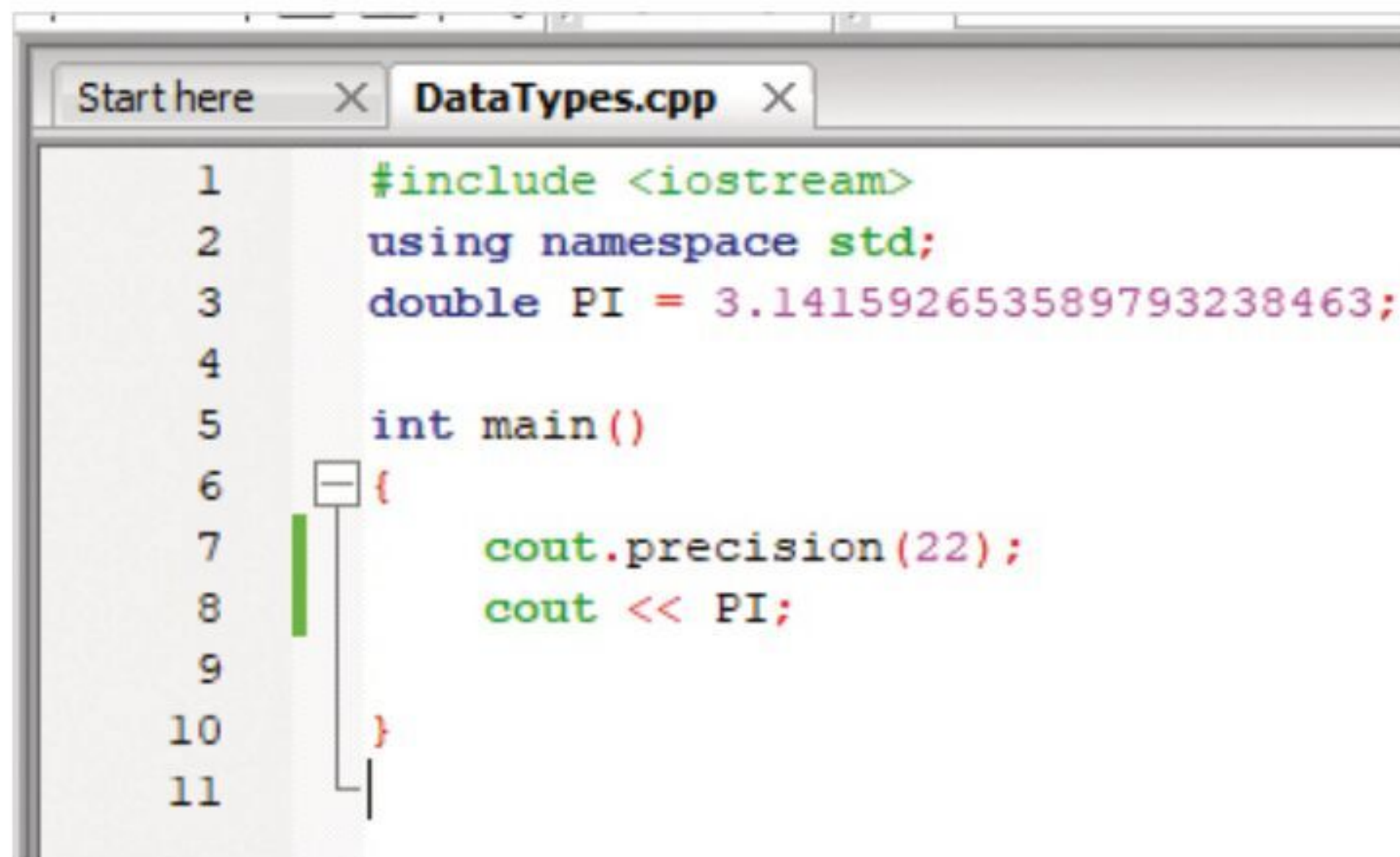
You can alter the code including the aforementioned `cout.precision()` function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

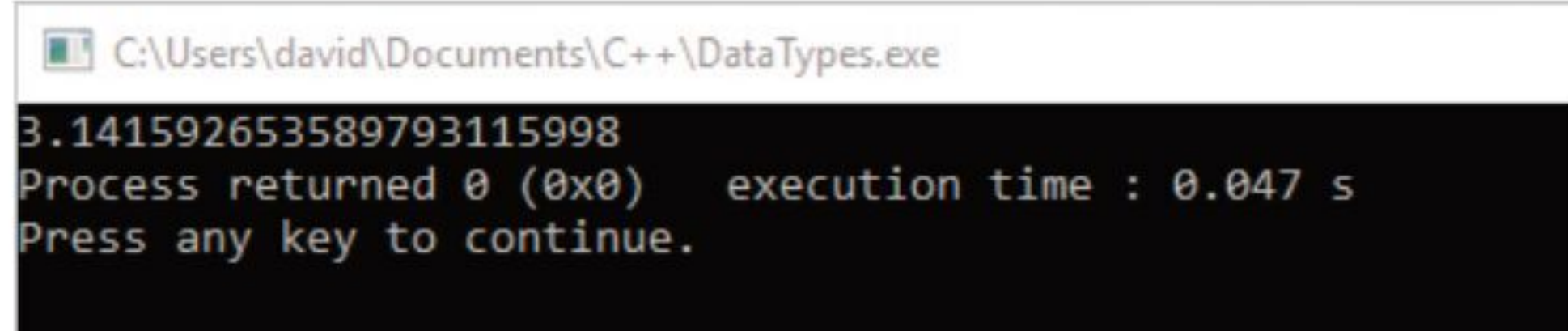
int main()
{
```



```
cout.precision(22);
cout << PI;
}
```



```
1 #include <iostream>
2 using namespace std;
3 double PI = 3.141592653589793238463;
4
5 int main()
6 {
7     cout.precision(22);
8     cout << PI;
9 }
10
11
```



```
C:\Users\david\Documents\C++\DataTypes.exe
3.141592653589793115998
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

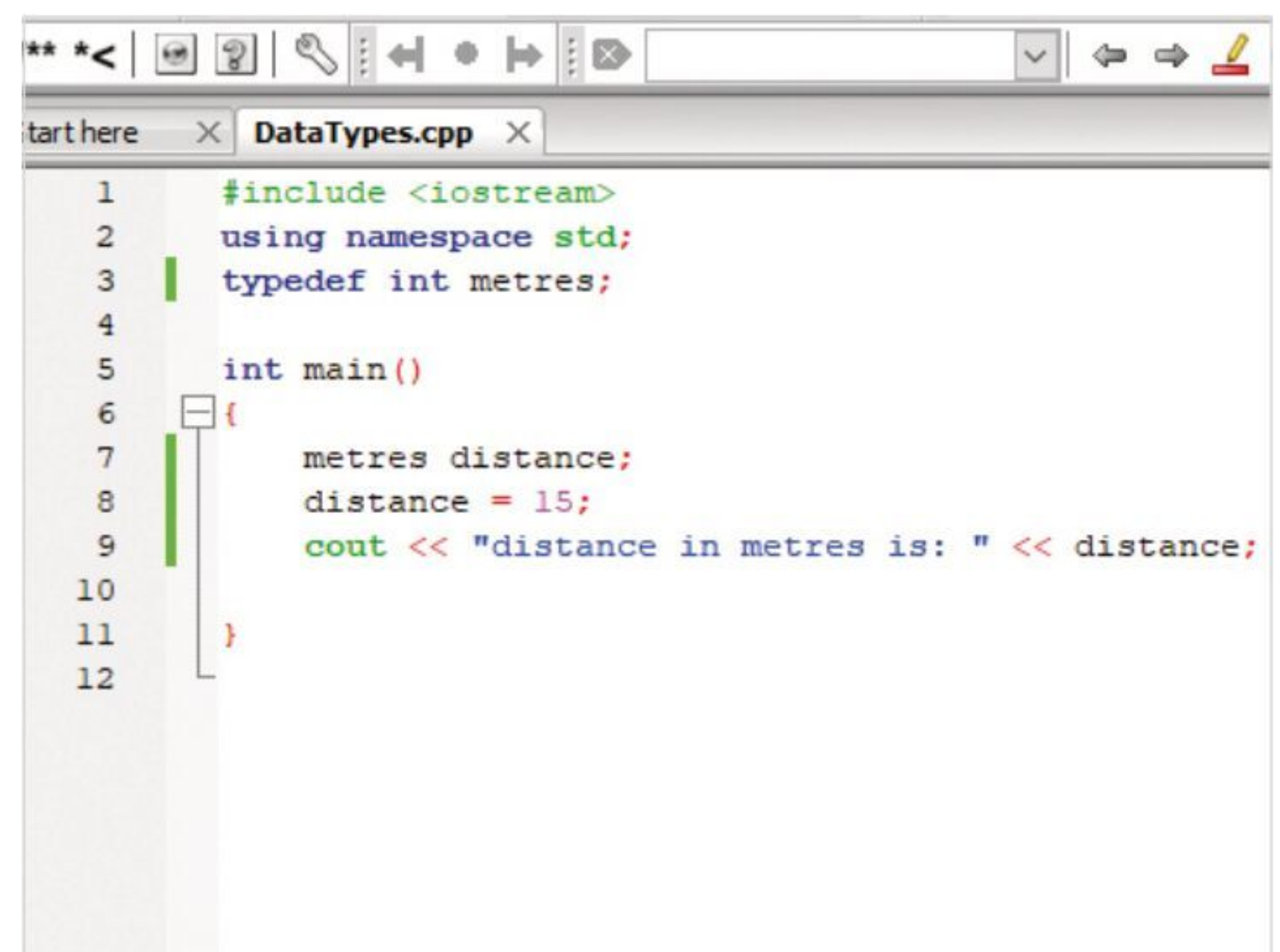
Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.



This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

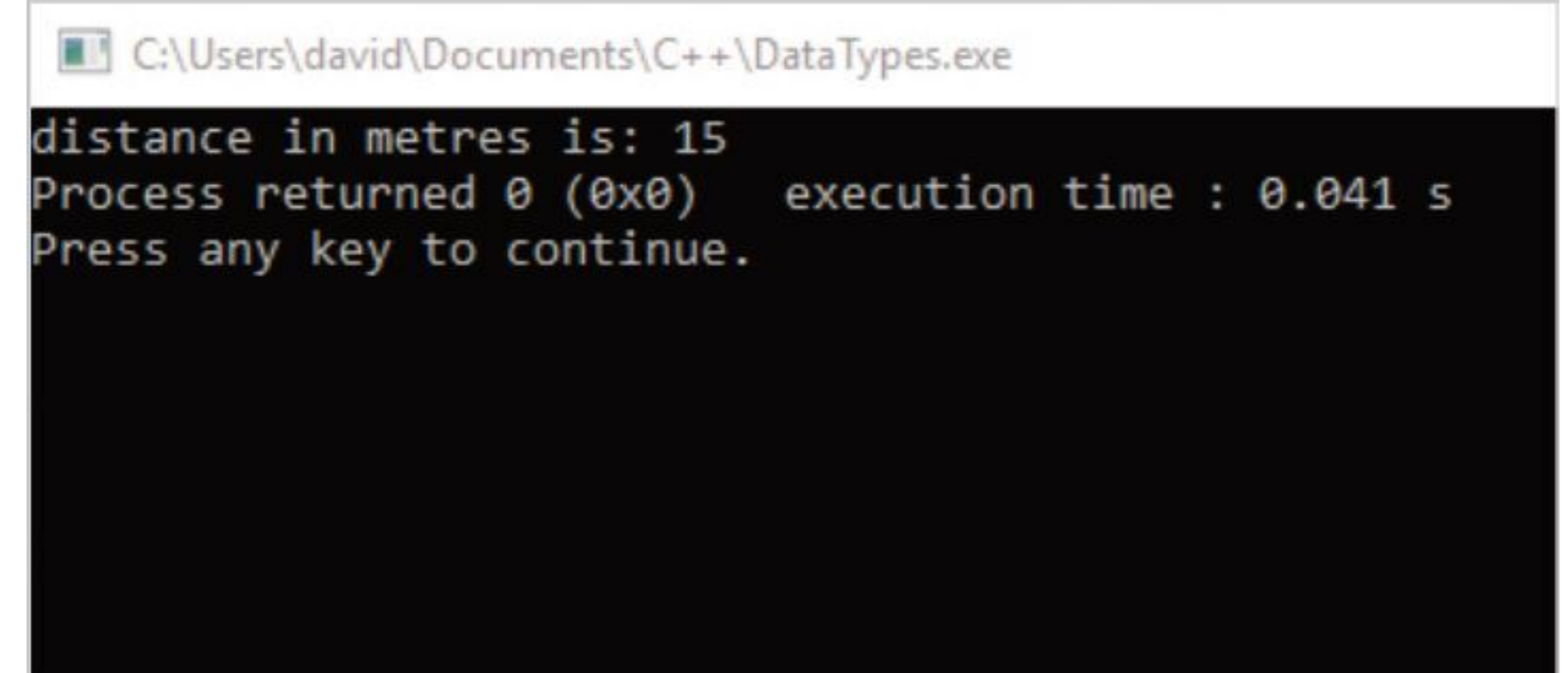
You can create your own data types, using an alias-like system called typedef. For example:



```
1 #include <iostream>
2 using namespace std;
3 typedef int metres;
4
5 int main()
6 {
7     metres distance;
8     distance = 15;
9     cout << "distance in metres is: " << distance;
10 }
11
12
```

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;
}
```



```
C:\Users\david\Documents\C++\DataTypes.exe
distance in metres is: 15
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're basically telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.



Strings

Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code 'Welcome' and assign that as a string to be called up wherever you like in the program.

STRING THEORY

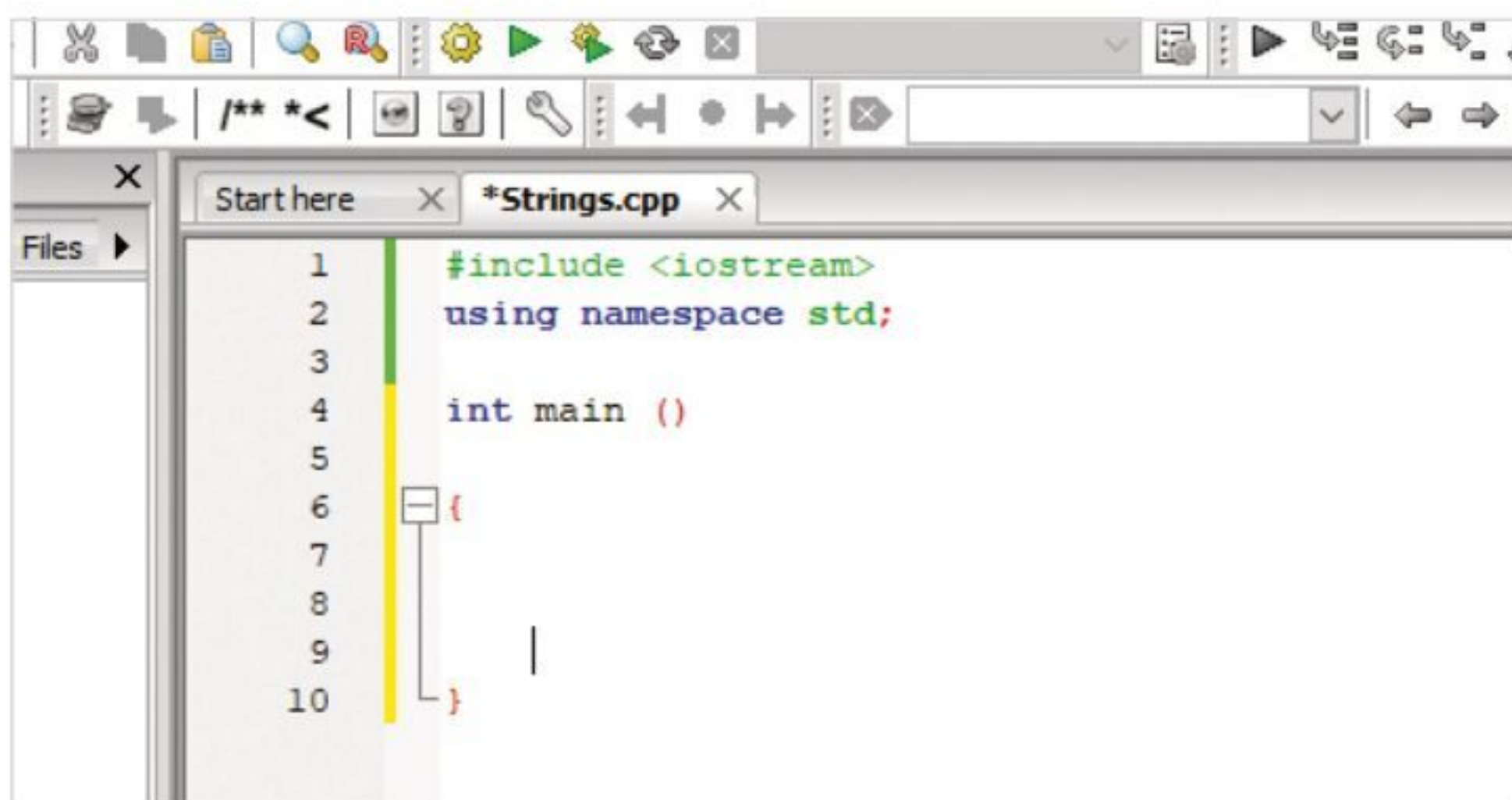
There are different ways in which you can create a string of characters, which historically are all carried over from the original C language, and are still supported by C++.

STEP 1

To create a string you use the char function. Open a new C++ file and begin with the usual header:

```
#include <iostream>
using namespace std;

int main ()
{
}
```

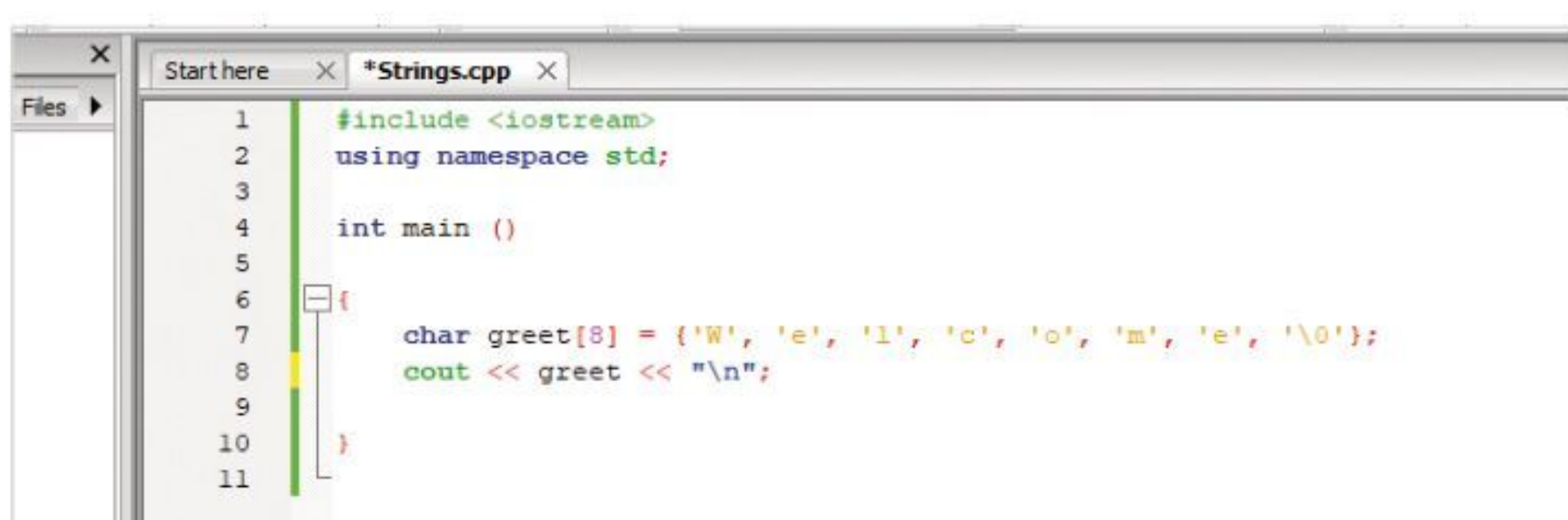


STEP 2

It's easy to confuse a string with an array. Here's an array, which can be terminated with a null character:

```
#include <iostream>
using namespace std;

int main ()
{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
    cout << greet << "\n";
}
```



STEP 3

Build and run the code, and 'Welcome' appears on the screen. While this is perfectly fine, it's not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn't need to be terminated like an array. The code can therefore be represented as:

```
#include <iostream>
using namespace std;

int main ()
{
    char greet[] = "Welcome";
    cout << greet << "\n";
}
```

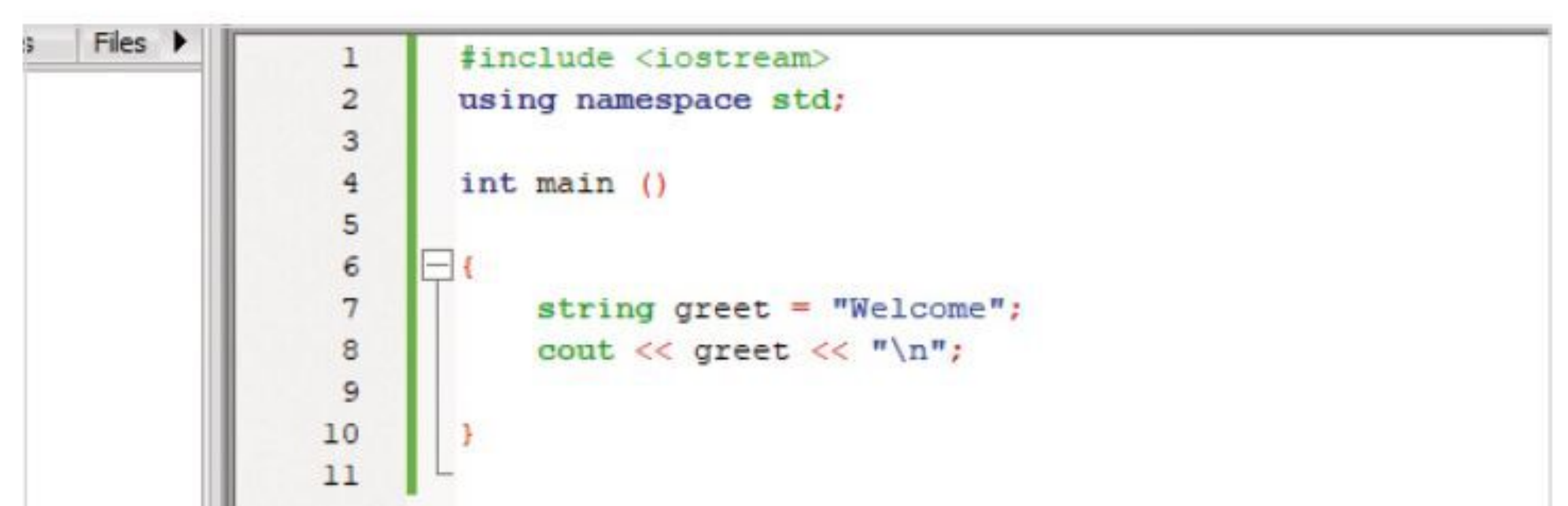


STEP 4

In C++ there's also a string function, which works in much the same way. Using the greeting code again, you can enter:

```
#include <iostream>
using namespace std;

int main ()
{
    string greet = "Welcome";
    cout << greet << "\n";
}
```

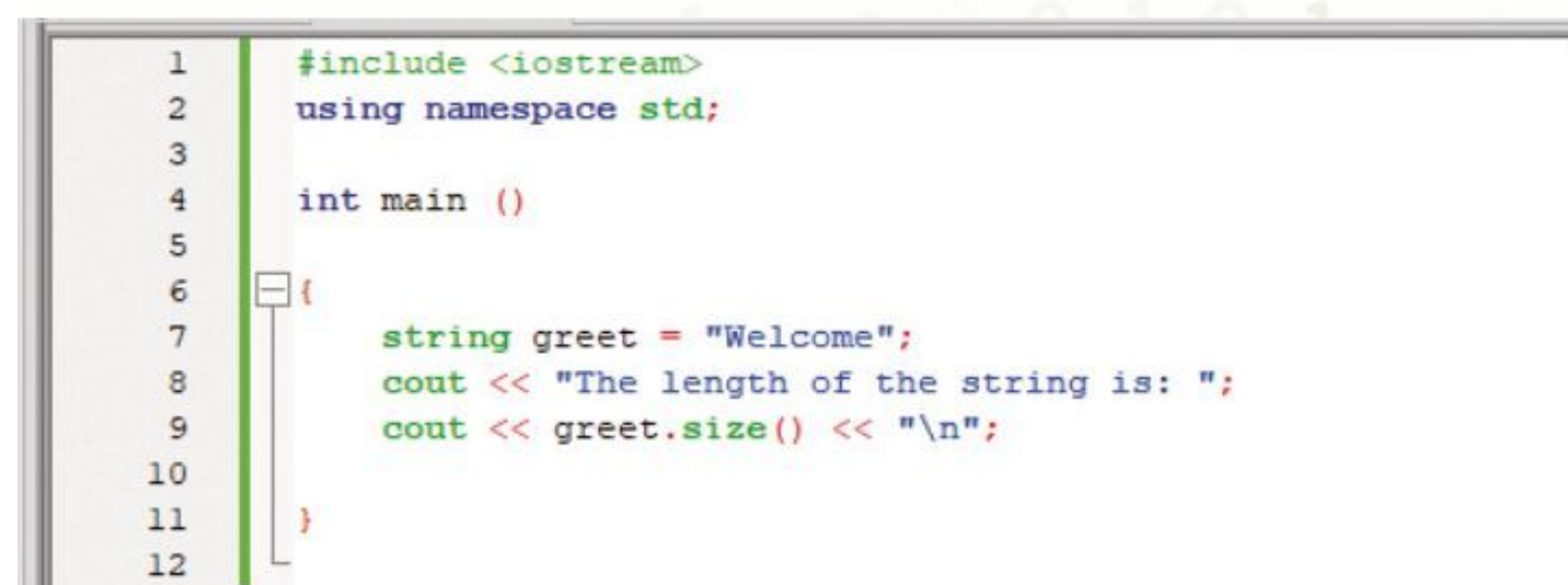


STEP 5

There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

```
#include <iostream>
using namespace std;

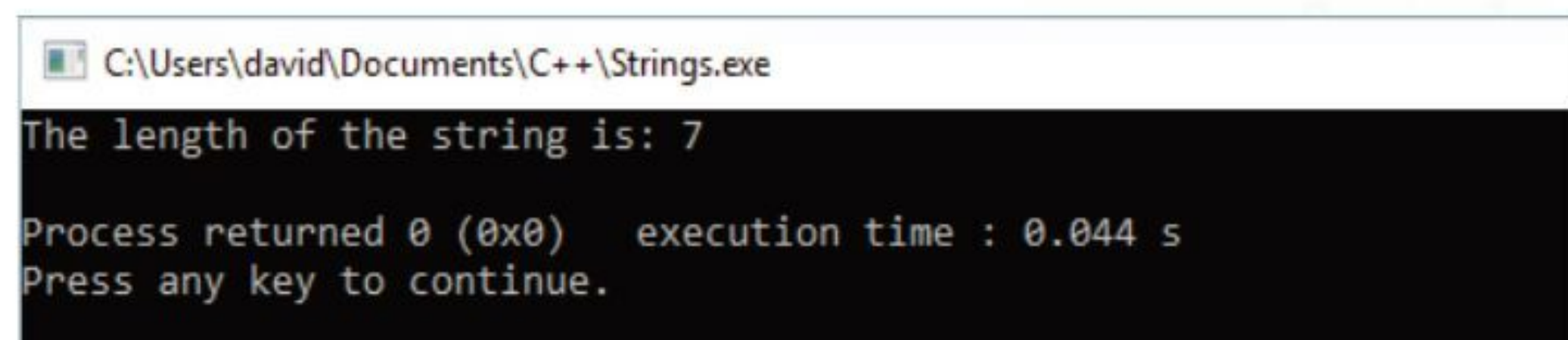
int main ()
{
    string greet = "Welcome";
    cout << "The length of the string is: ";
    cout << greet.size() << "\n";
}
```



```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      string greet = "Welcome";
7      cout << "The length of the string is: ";
8      cout << greet.size() << "\n";
9  }
```

STEP 6

You can see that we used `greet.size()` to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than `greet`, then you need to change the command to reflect this. It's always `stringname.operation`. Build and run the code to see the results.



```
C:\Users\david\Documents\C++\Strings.exe
The length of the string is: 7

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7

You can of course add strings together, or rather combine them to form longer strings:

```
#include <iostream>
using namespace std;

int main ()
{
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    cout << greet3 << "\n";
}
```



```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6      string greet1 = "Hello";
7      string greet2 = ", world!";
8      string greet3 = greet1 + greet2;
9
10     cout << greet3 << "\n";
11 }
12
13
14
```

STEP 8

Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created `int length`, which stores the result of `string.size()` and outputs it to the user:

```
#include <iostream>
using namespace std;

int main ()
{
    int length;
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    length = greet3.size();
    cout << "The length of the combined strings
is: " << length << "\n";
}
```

STEP 9

Using the available operations that come with the string function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

```
#include <iostream>
using namespace std;

int main ()
{
    string strg ("Here is a long sentence in a
string.");
    cout << strg << '\n';

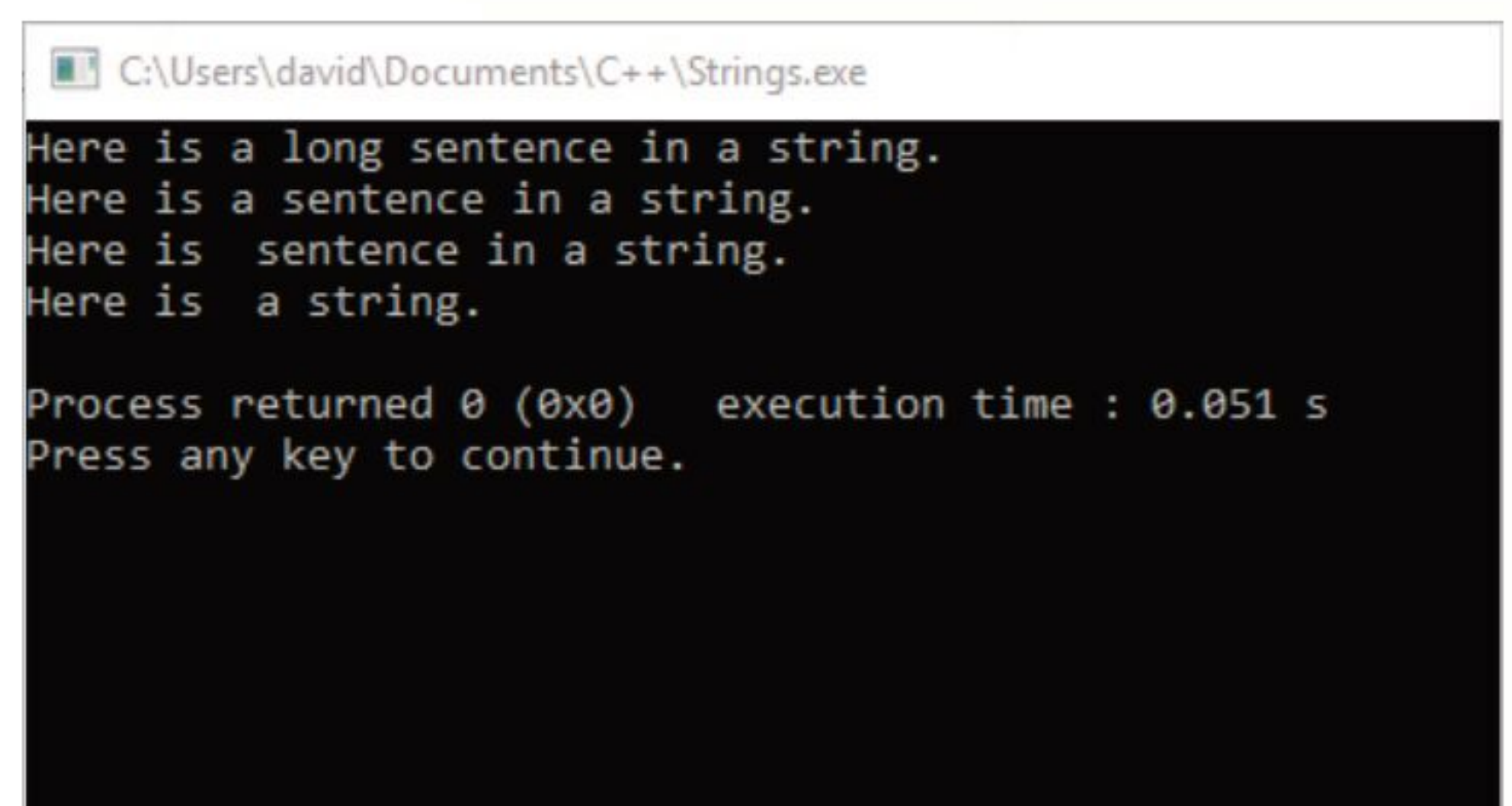
    strg.erase (10,5);
    cout << strg << '\n';

    strg.erase (strg.begin()+8);
    cout << strg << '\n';

    strg.erase (strg.begin()+9, strg.end()-9);
    cout << strg << '\n';
}
```

STEP 10

It's worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.



```
C:\Users\david\Documents\C++\Strings.exe
Here is a long sentence in a string.
Here is a sentence in a string.
Here is  sentence in a string.
Here is  a string.

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```




C++ Maths

Programming is mathematical in nature and as you might expect, there's plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who's implementing mathematical models into their code. It can be extremely complex or relatively simple.

C++ = MC²

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

STEP 1

C++'s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

```
#include <iostream>
using namespace std;

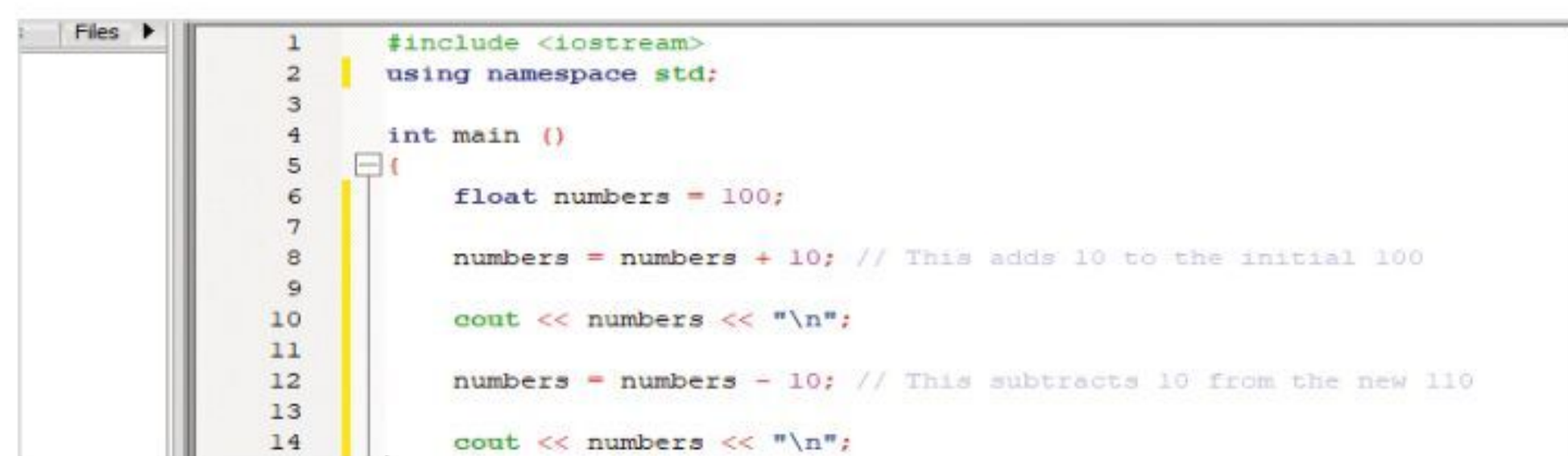
int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the
    initial 100

    cout << numbers << "\n";

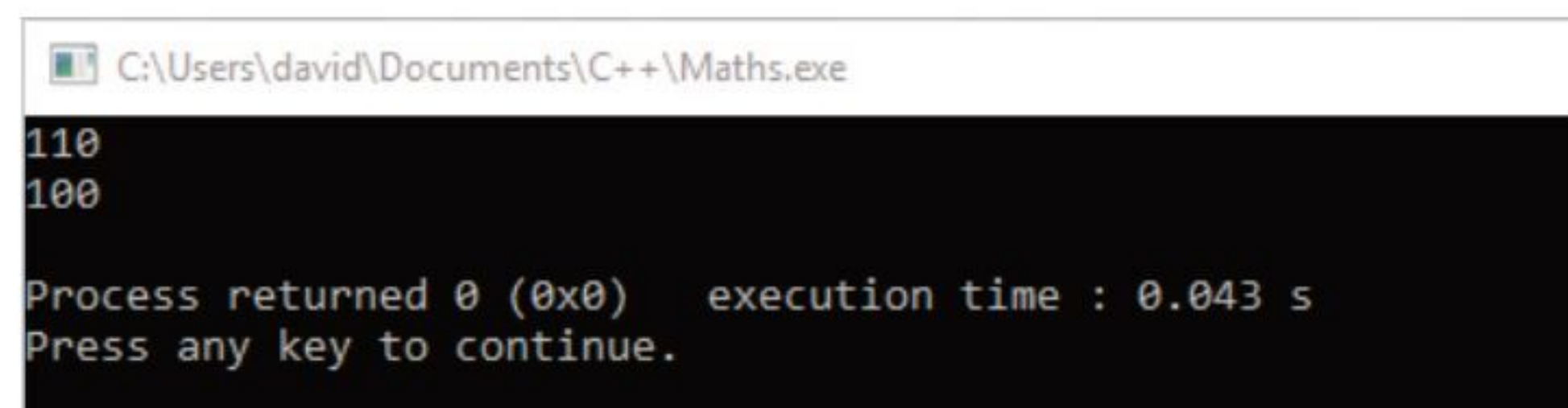
    numbers = numbers - 10; // This subtracts 10
    from the new 110

    cout << numbers << "\n";
}
```



STEP 2

While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.



STEP 3

Multiplication and division can be applied as such:

```
#include <iostream>
using namespace std;

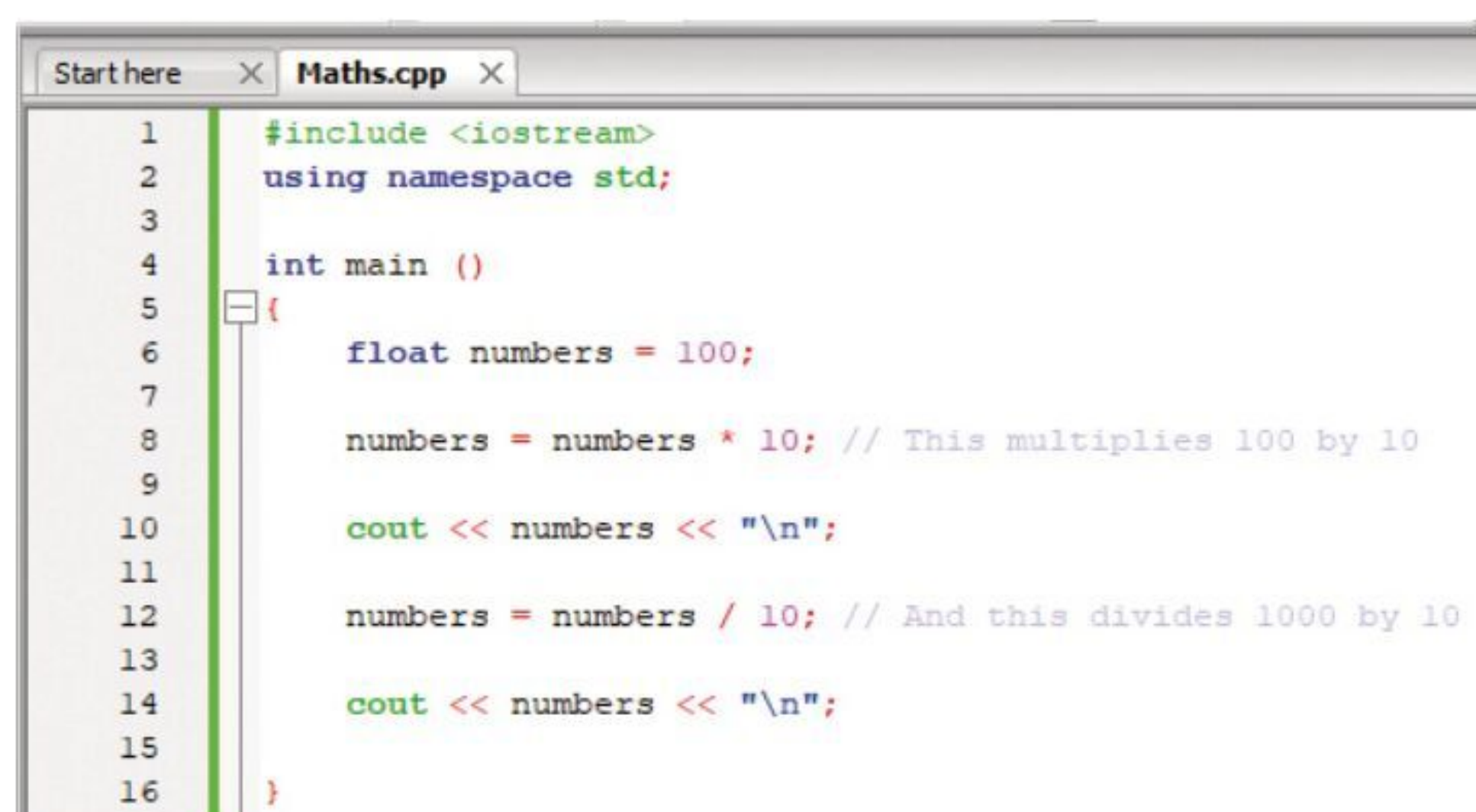
int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100
    by 10

    cout << numbers << "\n";

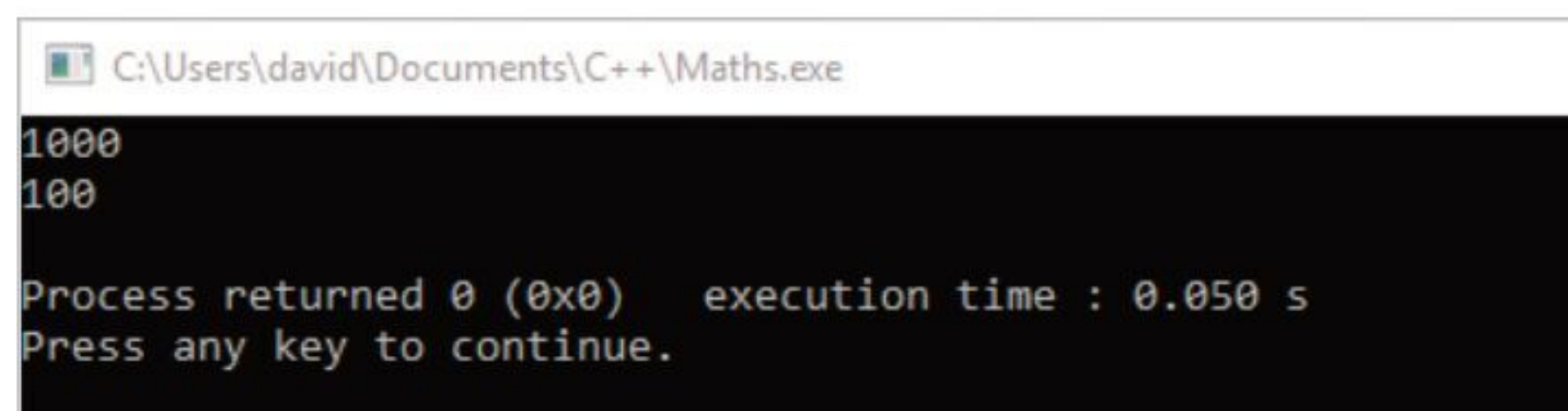
    numbers = numbers / 10; // And this divides
    1000 by 10

    cout << numbers << "\n";
}
```



STEP 4

Again, execute the simple code and see the results. While not particularly interesting, it's a start into C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.

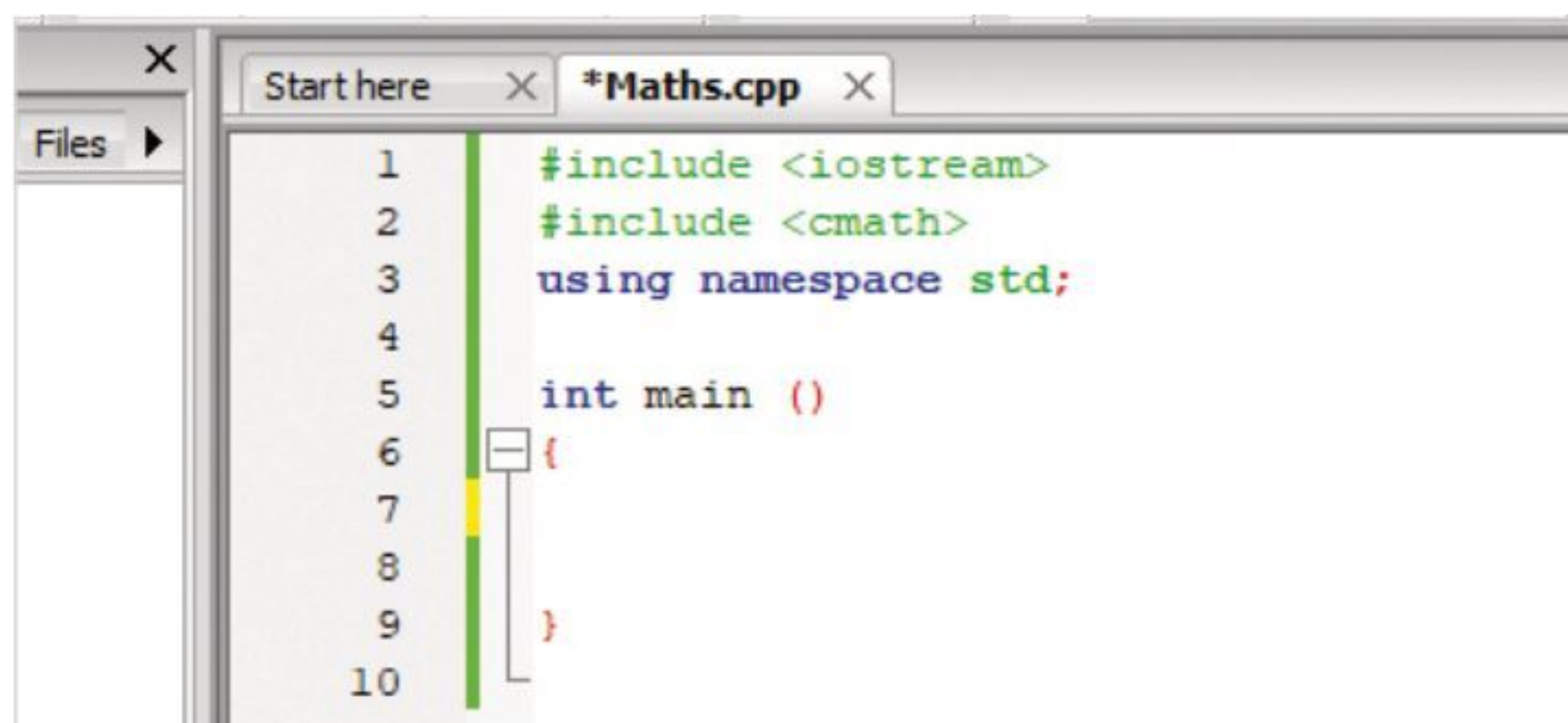


STEP 5

The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
}
```

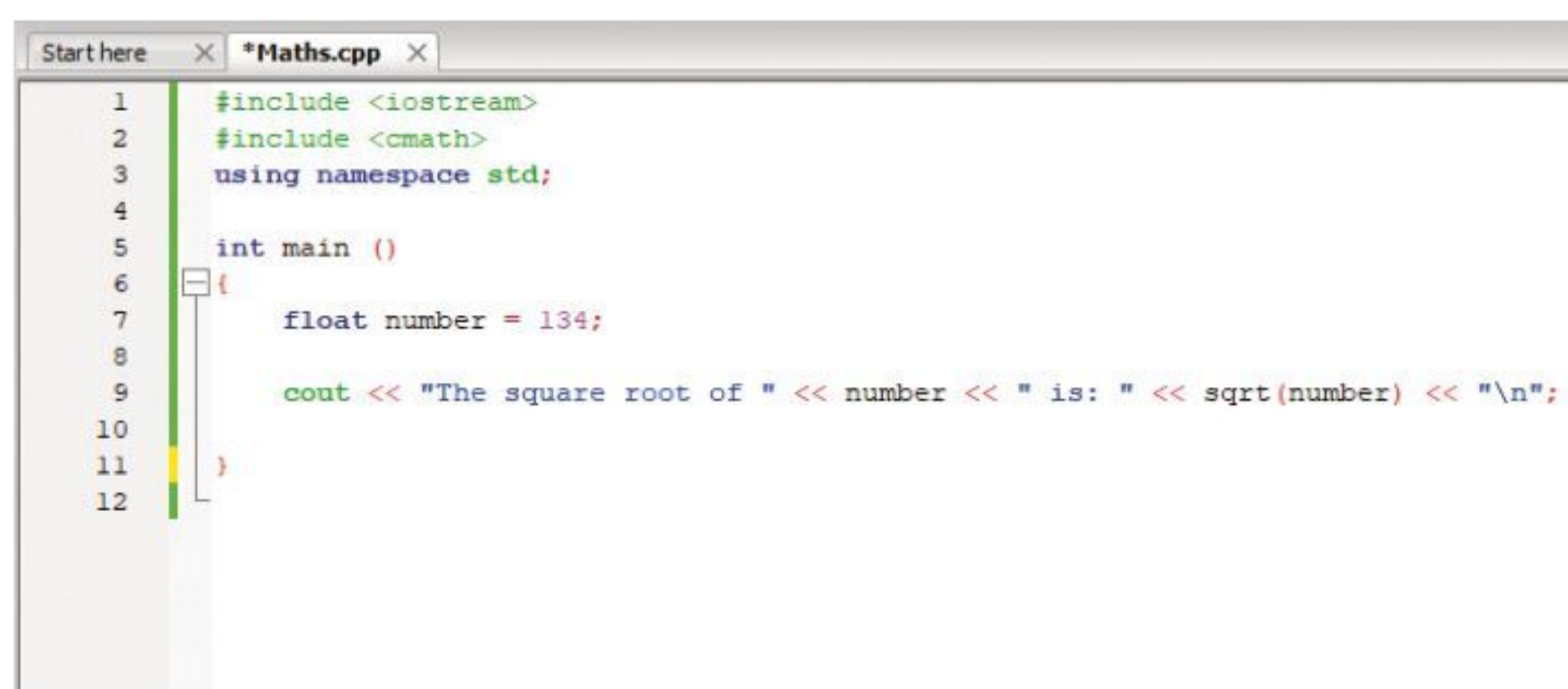
**STEP 6**

Start by getting the square root of a number:

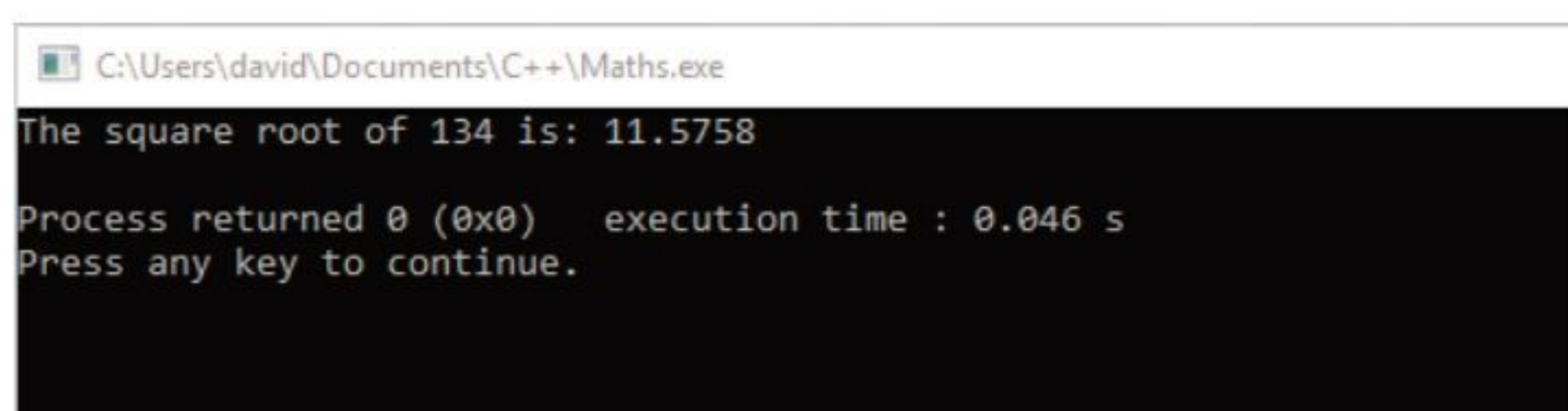
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << "The square root of " << number << "
    is: " << sqrt(number) << "\n";
}
```

**STEP 7**

Here we created a new float called number and used the sqrt(number) function to display the square root of 134, the value of the variable, number. Build and run the code, and your answer reads 11.5758.

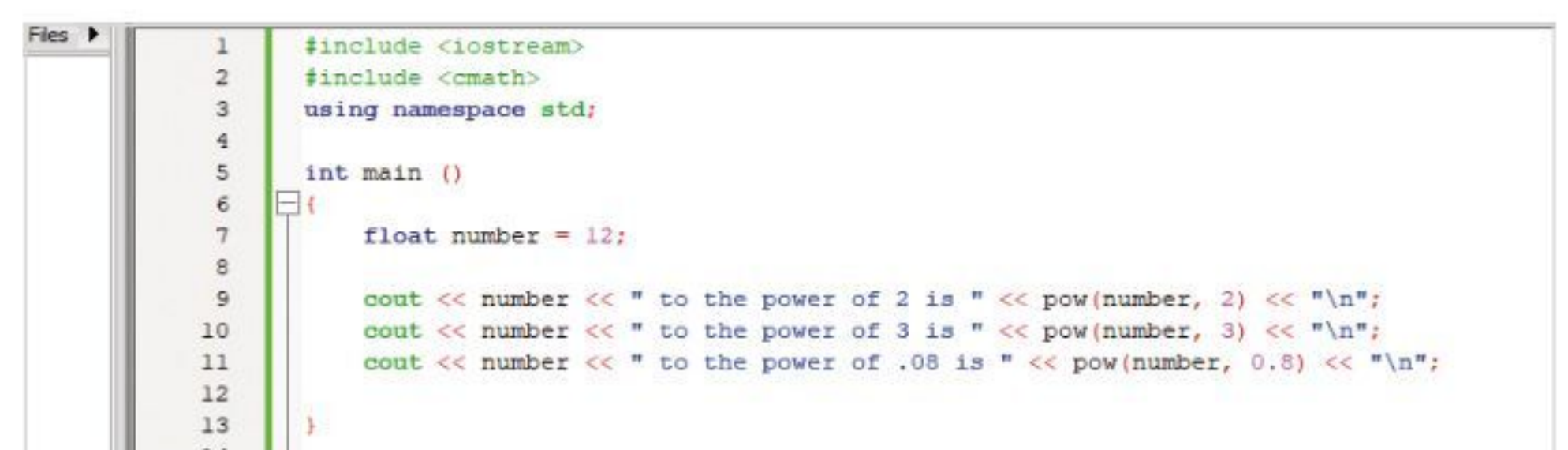
**STEP 8**

Calculating powers of numbers can be done with:

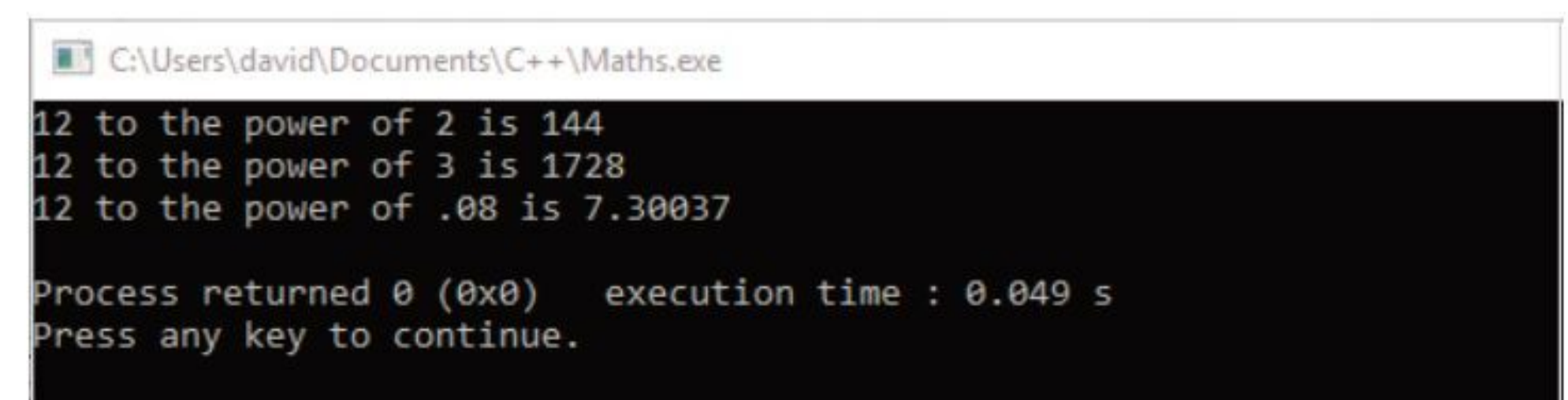
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << " to the power of 2 is " <<
    pow(number, 2) << "\n";
    cout << number << " to the power of 3 is " <<
    pow(number, 3) << "\n";
    cout << number << " to the power of .08 is "
    << pow(number, 0.8) << "\n";
}
```

**STEP 9**

Here we created a float called number with the value of 12, and the pow(variable, power) is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, pow(12, 2) outputs the same value as the first cout line in the code.

**STEP 10**

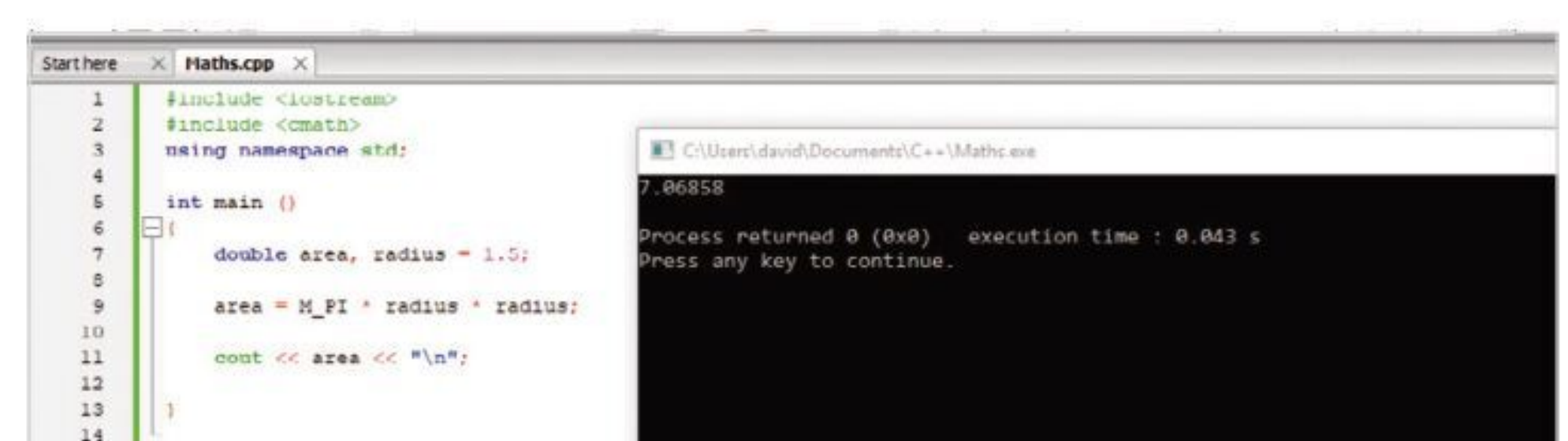
The value of Pi is also stored in the cmath header library. It can be called up with the M_PI function. Enter cout << M_PI; into the code and you get 3.14159; or you can use it to calculate:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << "\n";
}
```





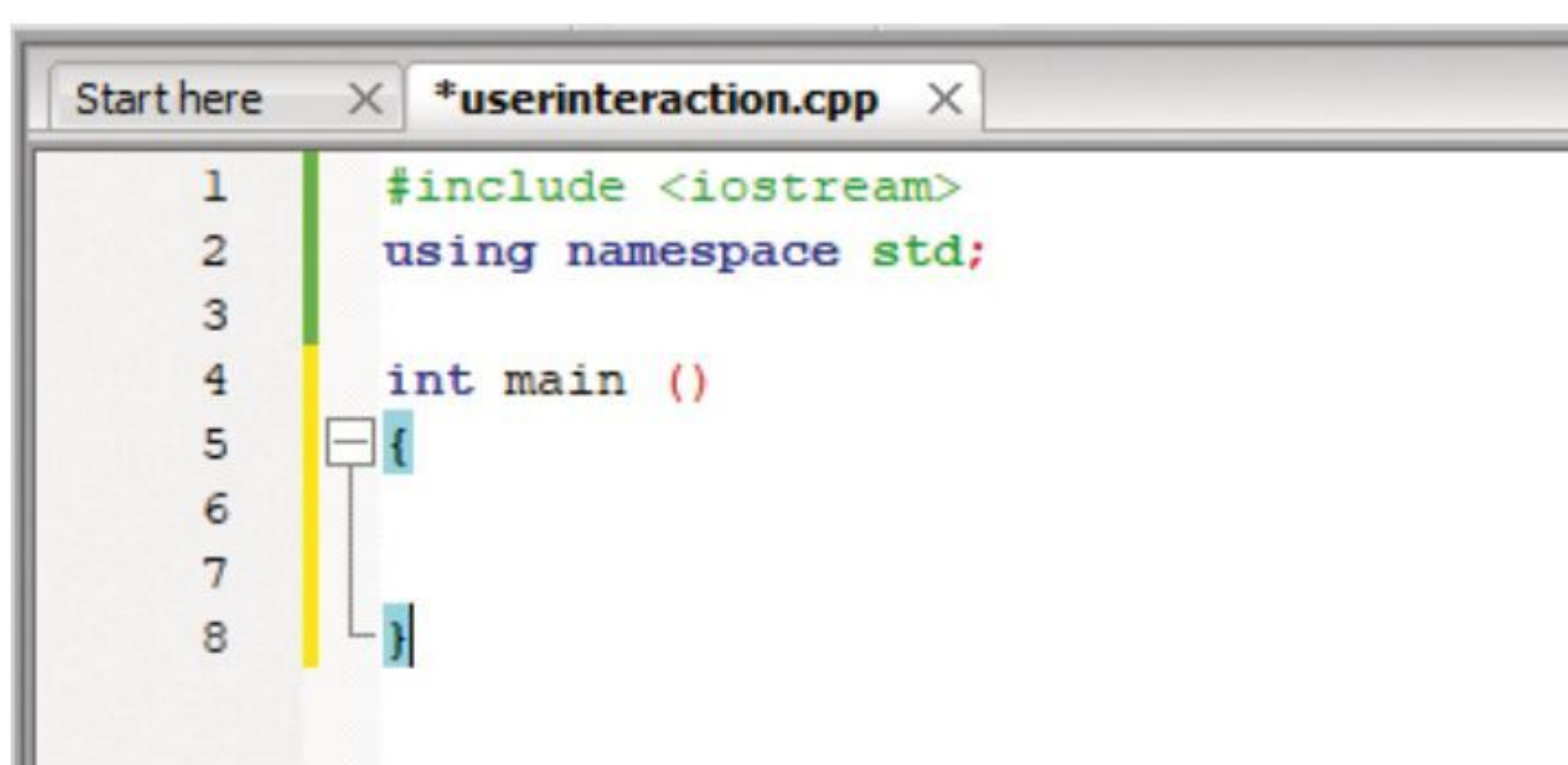
User Interaction

There's nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you're able to do much more than simply greet the user by name.

HELLO, DAVE

You have already used `cout`, the standard output stream, throughout our code. Now you're going to be using `cin`, the standard input stream, to prompt a user response.

STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it's ready to be used by the user. Start by creating a blank C++ file with headers.



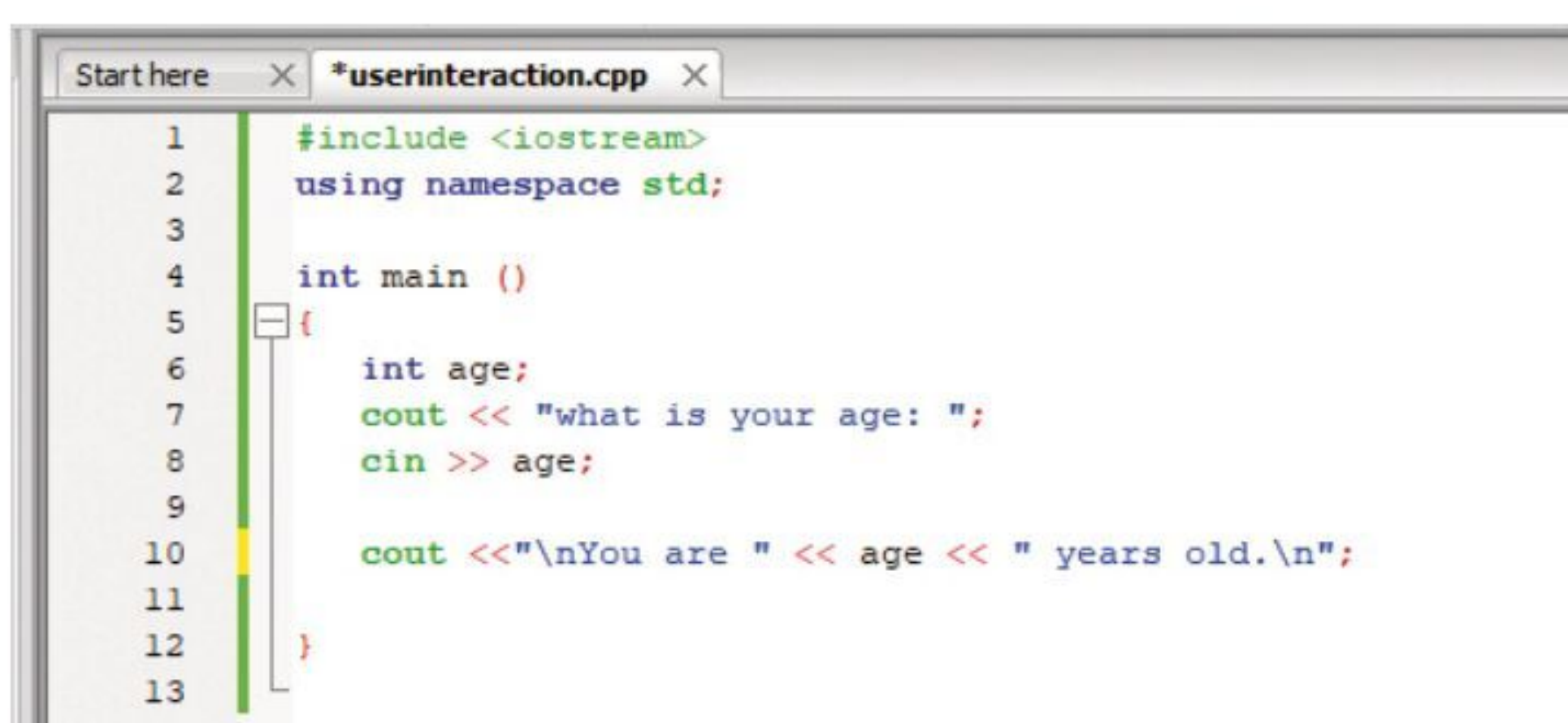
```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6
7
8 }
```

STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

```
#include <iostream>
using namespace std;

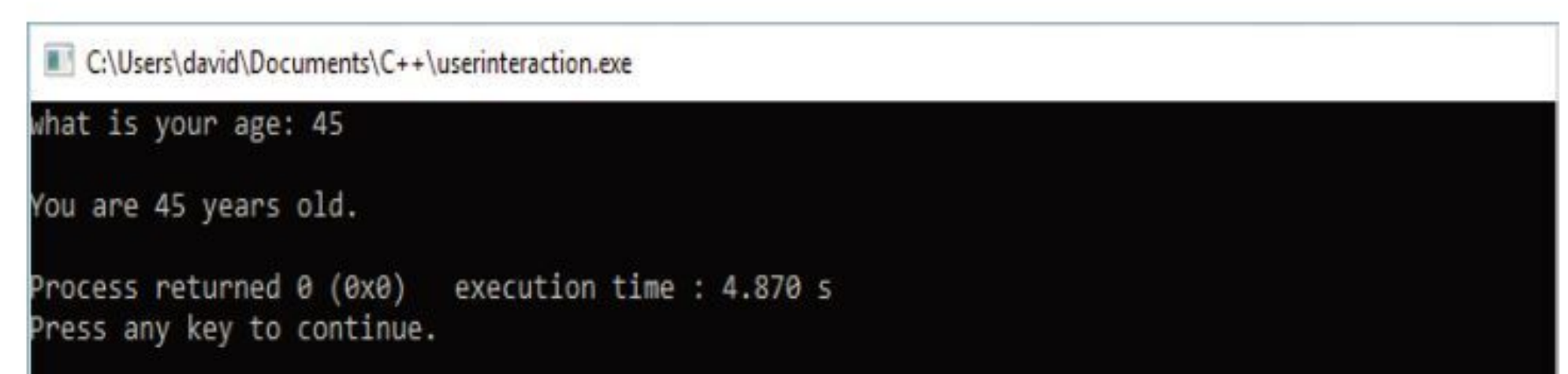
int main ()
{
    int age;
    cout << "what is your age: ";
    cin >> age;

    cout << "\nYou are " << age << " years old.\n";
}
```



```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     int age;
7     cout << "what is your age: ";
8     cin >> age;
9
10    cout << "\nYou are " << age << " years old.\n";
11
12 }
13
```

STEP 3 The `cin` command works in the opposite way from the `cout` command. With the first `cout` line you're outputting 'What is your age' to the screen, as indicated with the chevrons. `Cin` uses opposite facing chevrons, indicating an input. The input is put into the integer `age` and called up in the second `cout` command. Build and run the code.



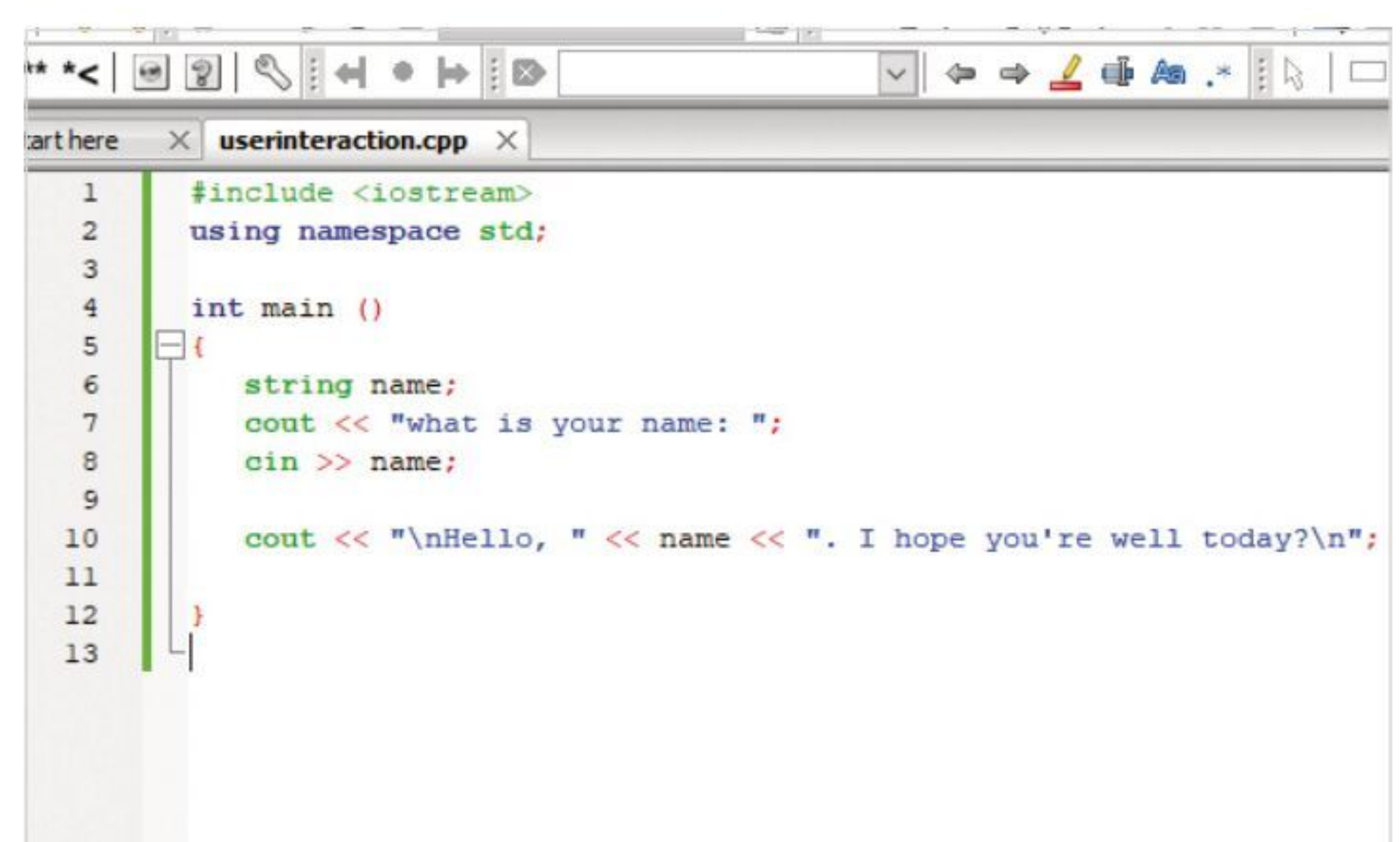
```
C:\Users\david\Documents\C++\userinteraction.exe
What is your age: 45
You are 45 years old.
Process returned 0 (0x0)   execution time : 4.870 s
Press any key to continue.
```

STEP 4 If you're asking a question, you need to store the input as a string; to ask the user their name, you would use:

```
#include <iostream>
using namespace std;

int main ()
{
    string name;
    cout << "what is your name: ";
    cin >> name;

    cout << "\nHello, " << name << ". I hope you're well today?\n";
}
```



```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     string name;
7     cout << "what is your name: ";
8     cin >> name;
9
10    cout << "\nHello, " << name << ". I hope you're well today?\n";
11
12 }
13
```


**STEP 5**

The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code.

```
C:\Users\david\Documents\C++\userinteraction.exe
what is your name: David
Hello, David. I hope you're well today?
Process returned 0 (0x0)   execution time : 2.153 s
Press any key to continue.
```

STEP 6

You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

```
#include <iostream>
using namespace std;

int main ()
{
    int num1, num2;

    cout << "Enter two whole numbers: ";
    cin >> num1 >> num2;

    cout << "you entered " << num1 << " and " <<
    num2 << "\n";
}
```

```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     int num1, num2;
7
8     cout << "Enter two whole numbers: ";
9     cin >> num1 >> num2;
10
11     cout << "you entered " << num1 << " and " << num2 << "\n";
12
13 }
```

STEP 7

Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

```
#include <iostream>
using namespace std;

int main ()
{
    float num1, num2;

    cout << "Enter two numbers: \n";
    cin >> num1 >> num2;

    cout << num1 << " + " << num2 << " is: " <<
    num1 + num2 << "\n";
}
```

```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     float num1, num2;
7
8     cout << "Enter two numbers: \n";
9     cin >> num1 >> num2;
10
11     cout << num1 << " + " << num2 << " is: " << num1 + num2 << "\n";
12
13
14 }
```

STEP 8

While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

```
#include <iostream>
using namespace std;

int main ()
{
    string mystr;
    cout << "Enter a sentence: \n";
    getline(cin, mystr);

    cout << "Your sentence is: " << mystr.size() <<
    " characters long.\n";
}
```

```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     string mystr;
7     cout << "Enter a sentence: \n";
8     getline(cin, mystr);
9
10     cout << "Your sentence is: " << mystr.size() << " characters long.\n";
11 }
```

STEP 9

Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.

```
C:\Users\david\Documents\C++\userinteraction.exe
Enter a sentence:
BDM Publications Python and C++ for Beginners
Your sentence is: 45 characters long.
Process returned 0 (0x0)   execution time : 27.054 s
Press any key to continue.
```

STEP 10

Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

```
#include <iostream>
using namespace std;

int main ()
{
    string name;
    cout << "Enter your full name: \n";
    getline(cin, name);

    cout << "\nHello, " << name << "\n";
}
```

```
1 #include <iostream>
2 using namespace std;
3
4 int main ()
5 {
6     string name;
7     cout << "Enter your full name: \n";
8     getline(cin, name);
9
10     cout << "\nHello, " << name << "\n";
11 }
```




Introducing Python

Python is one of the most respected and used programming languages in the world. It's designed to help beginners get into code, but also has some amazingly powerful features that are utilised by data scientists and engineers around the world.

You don't need to look too far to Python at work. It's used throughout the Internet, and helps drive some of the biggest projects and companies in a multitude of industries.

In this section, we will look at what you will need to get started with Python. First steps into a new learning experience are always the hardest, but we're here to help.



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.

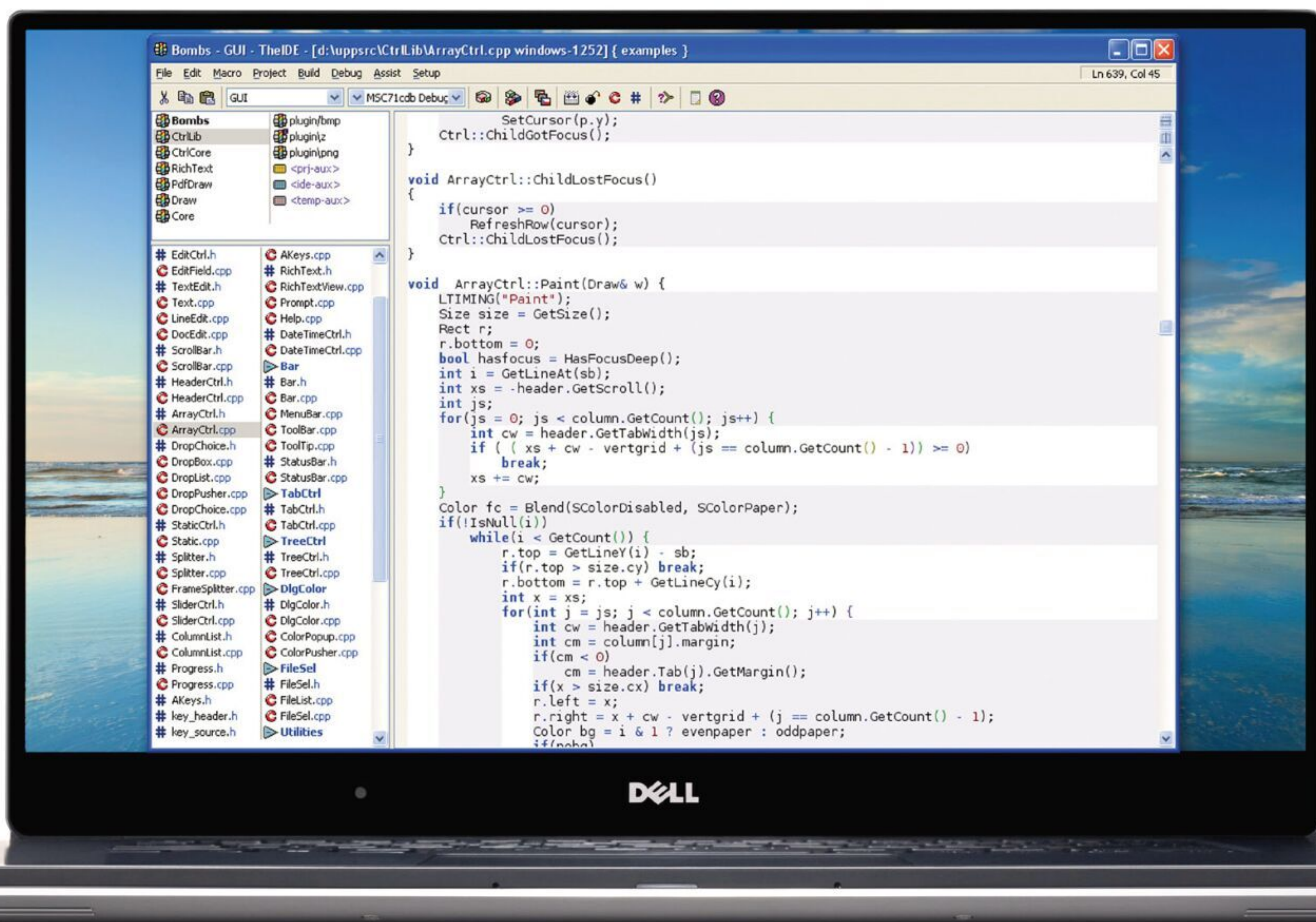
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra high-definition graphics and open world, multi-player online gaming. However, forty-something 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                 [] { } );
10            Object ret = m.invoke( null, null );
11            System.out.println(
12                "Invoked static method: " + args[1]
13                + " of class: " + args[0]
14                + " with no args\nResults: " + ret );
15        } catch ( ClassNotFoundException e ) {
16            // Class.forName( ) can't find the class
17        } catch ( NoSuchMethodException e2 ) {
18            // that method doesn't exist
19        } catch ( IllegalAccessException e3 ) {
20            // we don't have permission to invoke that
21            // method
22        } catch ( InvocationTargetException e4 ) {
23            // an exception occurred while invoking that
24            // method
25            System.out.println(
26                "Method threw an: " + e4.
27                getTargetException( ) );
28        }
29    }
30 }
```



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
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.



What Can You Do with Python?

Python is an open-source, object-oriented programming language that's simple to understand and write with, yet also powerful and extremely malleable. It's these characteristics that help make it such an important language to learn.

Python's ability to create highly readable code, within a small set of instructions, has a considerable impact on our modern digital world. From being an ideal first programmers' choice through to being able to create interactive stories and games; from scientific applications through to artificial Intelligence and web-based applications, the only limit to Python is the imagination of the person coding in it.

It's Python's malleable design that makes it an ideal language for many different situations and roles. Even certain aspects of the coding world that require more efficient code still use Python. For example, NASA utilises Python both as a stand-alone language and as a bridge between other programming languages. This way, NASA scientists and engineers are able to get to the data they need without having to cross multiple language barriers; Python fills the gaps and provides the means to get the job done.

You'll find lots of examples of this, where Python is acting behind the scenes. This is why it's such an important language to learn.



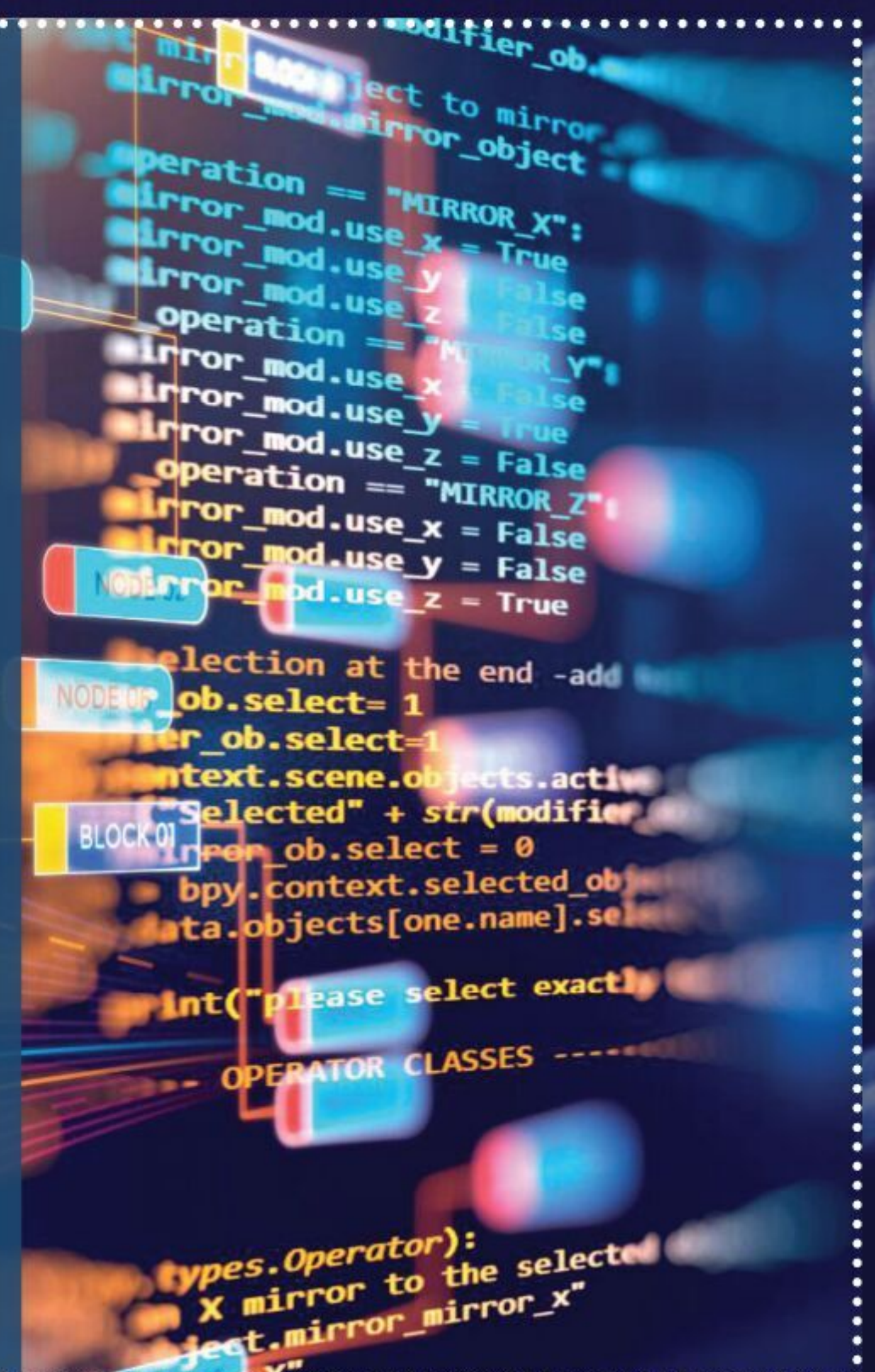
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

BIG DATA

Big data is a buzzword you're likely to have come across in the last couple of years. Basically, it means extremely large data sets that are available for analysis to reveal patterns, trends and interactions between humans, society and technology. Of course, it's not just limited to those areas, big data is currently being used in a variety of industries, from social media to health and welfare, engineering to space exploration and beyond.

Python plays a substantial role in the world of big data. It's extensively used to analyse huge chunks of the available big data and extract specific information based on what the user/company requires from the wealth of numbers present. Thanks to an impressive set of data processing libraries, Python makes the act of getting to the data, amongst the numbers, that counts and presenting it in a fashion that's readable and useable for humans.

There are countless libraries and freely available modules that enable fast, secure and more importantly, accurate processing of data from the likes of supercomputing clusters. For example, CERN uses a custom Python module to help analyse the 600 million collisions per second that the Large Hadron Collider (LHC) produces. A different language handles the raw data, but Python is present to help sift through the data so scientists can get to the content they want without the need to learn a far more complex programming language.



ARTIFICIAL INTELLIGENCE

Artificial Intelligence and Machine Learning are two of the most groundbreaking aspects of modern computing. AI is the umbrella term used for any computing process wherein the machine is doing something intelligent, working and reacting in similar ways to humans. Machine Learning is a subset of AI and provides the overall AI system with the ability to learn from its experiences.

However, AI isn't simply the creation of autonomous robots intent on wiping out human civilisation. Indeed, AI can be found in a variety of day-to-day computing applications where the 'machine', or more accurately the code, needs to learn from the actions of some form of input and anticipate what the input is likely to require, or do, next.

This model can be applied to Facebook, Google, Twitter, Instagram and so on. Have you ever looked up a celebrity on Instagram and then discovered that your searches within other social media platforms are now specifically targeted toward similar celebrities? This is a prime example of using AI in targeted advertising and behind the code and algorithms that predict what you're looking for, is Python.

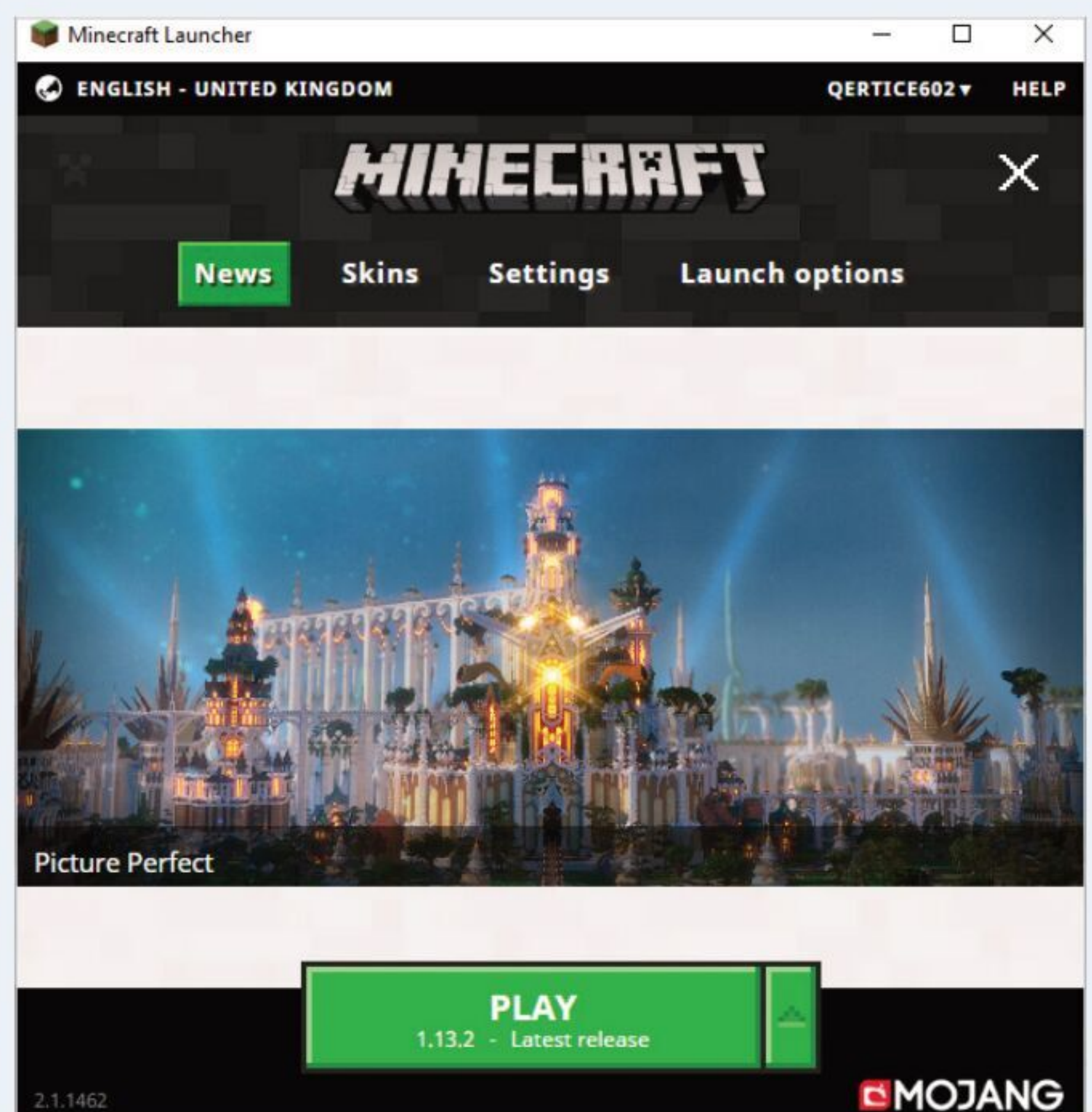
Spotify, for example, uses Python based code, among other things, to analyse your musical habits and offer playlists based on what you've listened to in the past. It's all clever stuff and moving forward, Python is at the forefront of the way the Internet will work in the future.



WEB DEVELOPMENT

Web development has moved on considerably since the early days of HTML scripting in a limited text editor. The many frameworks and web management services available now means that building a page has become increasingly complex.

With Python, the web developer has the ability to create dynamic and highly secure web apps, enabling interaction with other web services and apps such as Instagram and Pinterest. Python also allows the collection of data from other websites and even apps built within other websites.



GAMING

Although you won't find too many triple-A rated games coded using Python, you may be surprised to learn that Python is used as an extra on many of the high-ranking modern games.

The main use of Python in gaming comes in the form of scripting, where a Python script can add customisations to the core game engine. Many map editors are Python compatible and you will also come across it if you build any mods for games, such as The Sims.

A lot of the online, MMORPG (Massively Multiplayer Online Role-Playing Game) games available utilise Python as a companion language for the server-side elements. These include: code to search for potential cheating, load balancing across the game's servers, player skill matchmaking and to check whether the player's client-side game matches the server's versions. There's also a Python module that can be included in a Minecraft server, enabling the server admin to add blocks, send messages and automate a lot of the background complexities of the game.

PYTHON EVERYWHERE

As you can see, Python is quite a versatile programming language. By learning Python, you are creating a well-rounded skillset that's able to take you into the next generation of computing, either professionally or simply as a hobbyist.

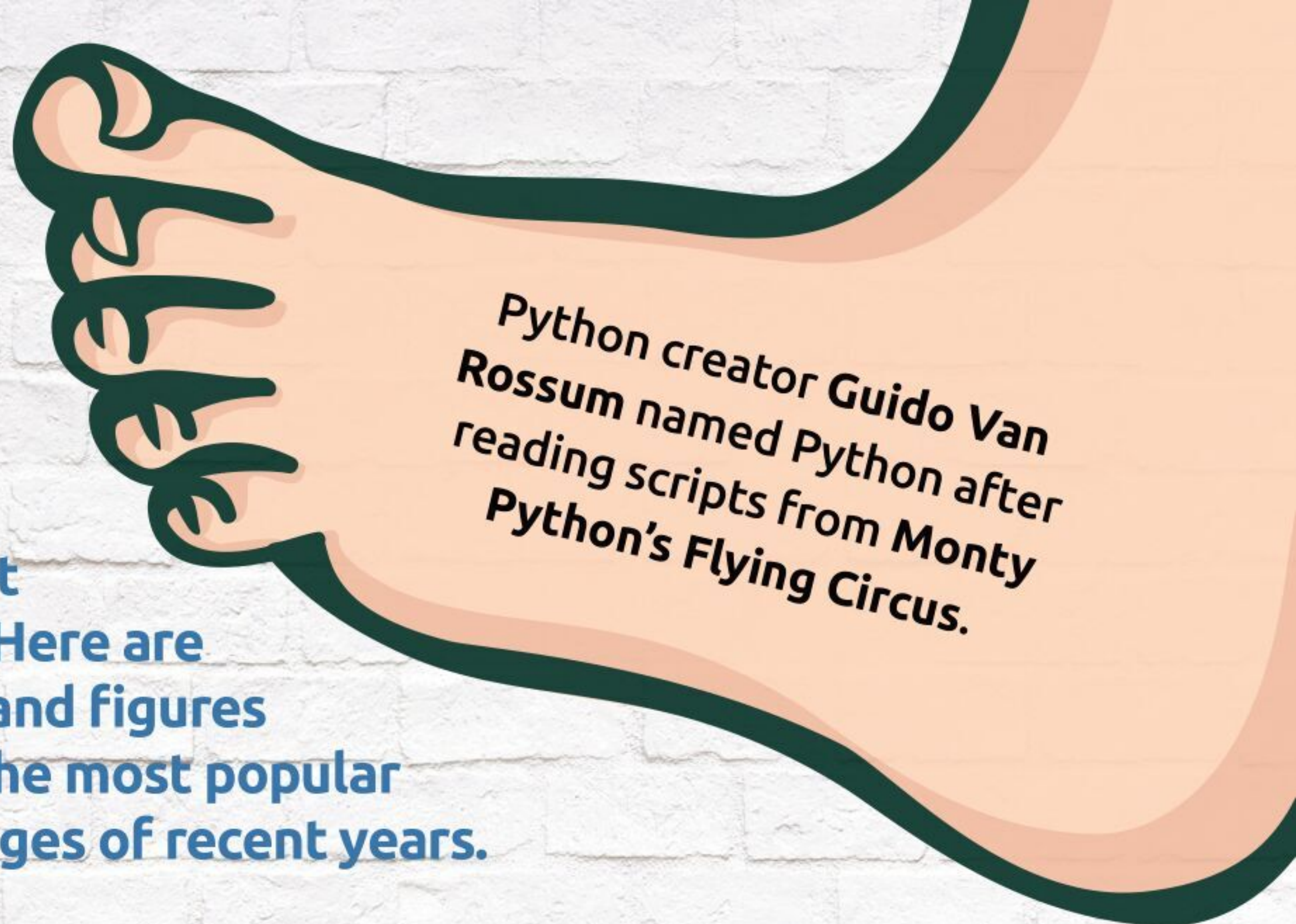
Whatever route you decide to take on your coding journey, you will do well to have Python in your corner.





PYTHON is IN NUMBERS

There's a lot to like about Python, but don't just take our word for it. Here are some amazing facts and figures surrounding one of the most popular programming languages of recent years.



Alexa, Amazon's Virtual Personal Assistant, uses Python to help with speech recognition.



.....
**PYTHON AND
LINUX SKILLS
ARE THE THIRD
MOST POPULAR
I.T. SKILLS IN
THE UK.**



Data analysis and Machine Learning are the two most used Python examples.



As of the end of 2018, Python was the most discussed language on the Internet.



Disney Pixar uses Python in its Renderman software to operate between other graphics packages.



OVER 75% OF RECOMMENDED CONTENT FROM NETFLIX IS GENERATED FROM MACHINE LEARNING – CODED BY PYTHON.

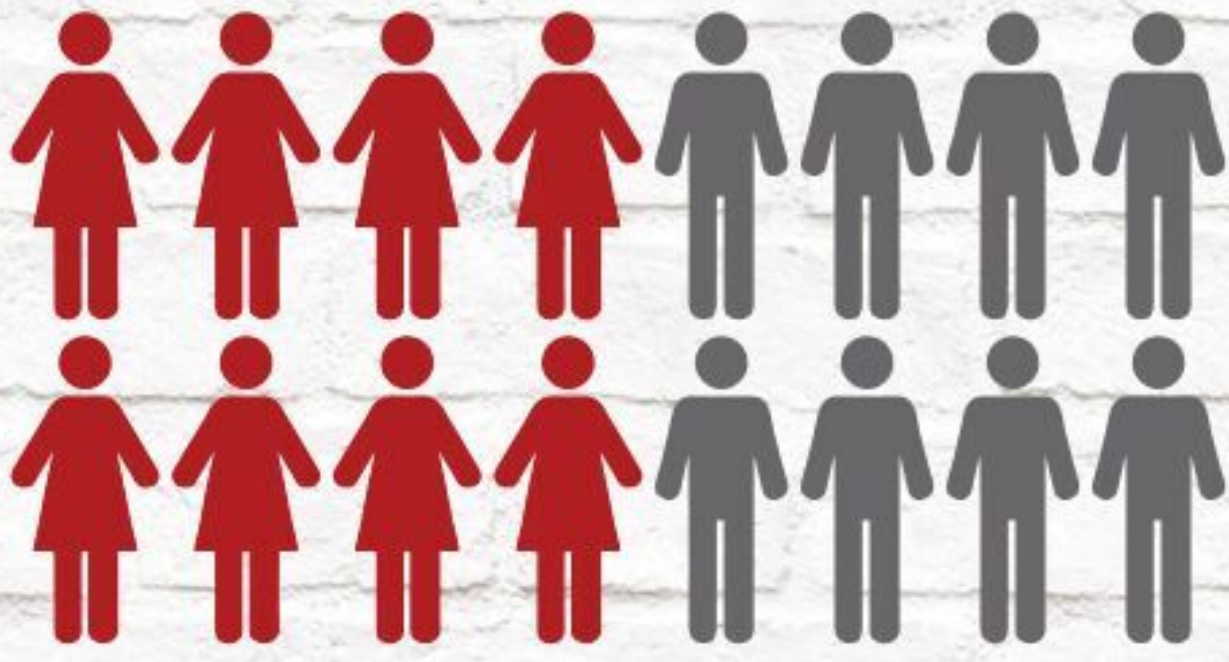


90% OF ALL FACEBOOK POSTS ARE FILTERED THROUGH PYTHON-CODED MACHINE LEARNING.



IT'S ESTIMATED THAT OVER 75% OF NASA'S WORKFLOW AUTOMATION SYSTEMS ON BOARD THE I.S.S. USE PYTHON.

16,000



There are over 16,000 Python jobs posted every six months in the UK.

PYTHON SKILL-BASED POSITIONS ARE THE

16th

MOST SOUGHT-AFTER JOBS IN THE UK.



Python Data Science is thought to become the most sought-after job in the coming years.



Google is the top company for hiring Python developers, closely followed by Microsoft.



Data Science, Blockchain and Machine Learning are the fastest growing Python coding skills.



New York and San Francisco are the top Python developer cities in the world.



Python developers enjoy an average salary of

£60,000

95%

95% OF ALL BEGINNER CODERS START WITH AND STILL USE, PYTHON AS THEIR PRIMARY OR SECONDARY LANGUAGE.

75%

75% OF ALL PYTHON DEVELOPERS USE PYTHON 3, WHEREAS 25% STILL USE THE OUTDATED PYTHON 2 VERSION.

79%

79% OF ALL PROGRAMMERS USE PYTHON REGULARLY, 21% USE IT AS A SECONDARY LANGUAGE.

49%

49% OF WINDOWS 10 DEVELOPERS USE PYTHON 3 AS THEIR MAIN 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

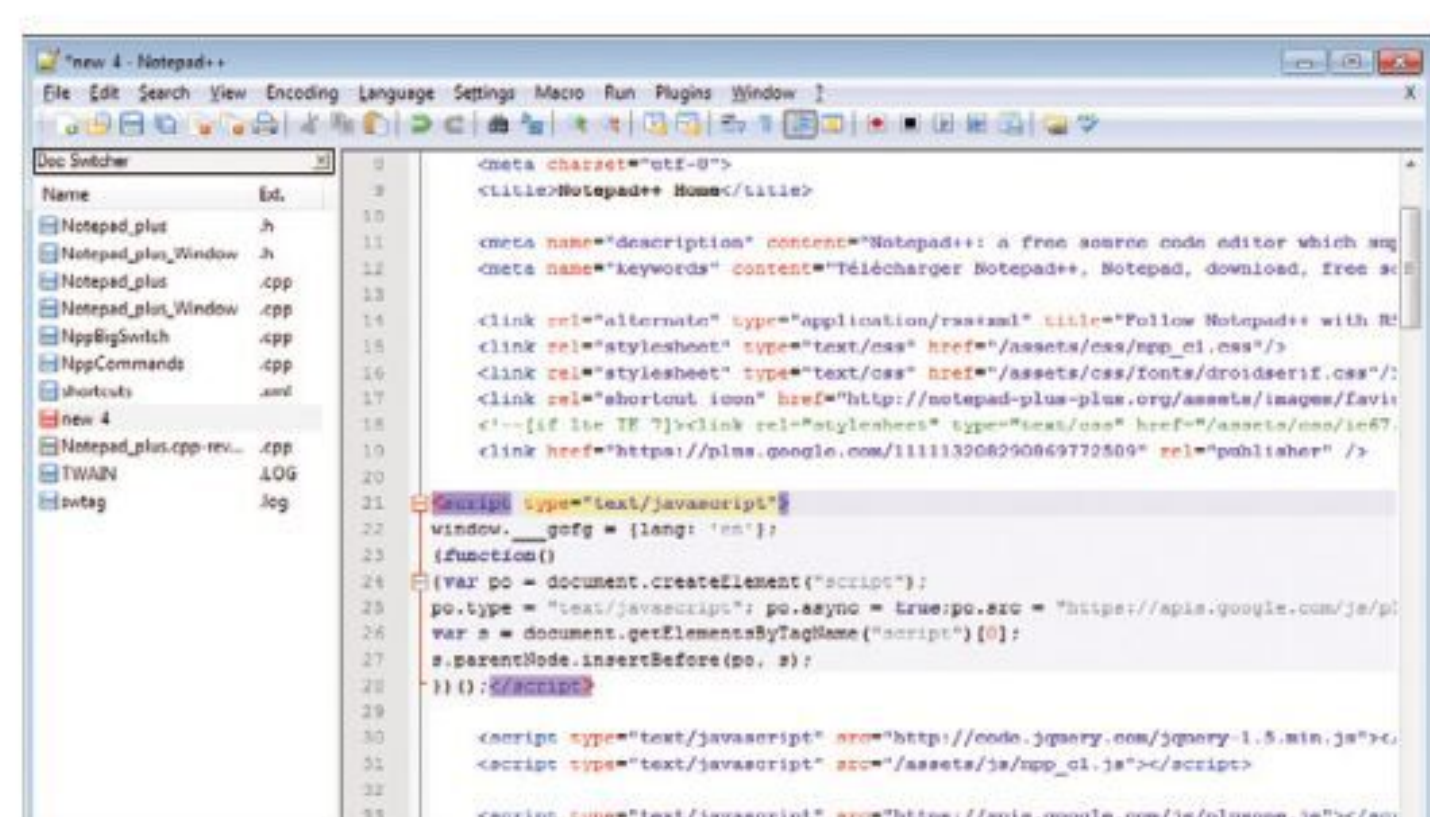
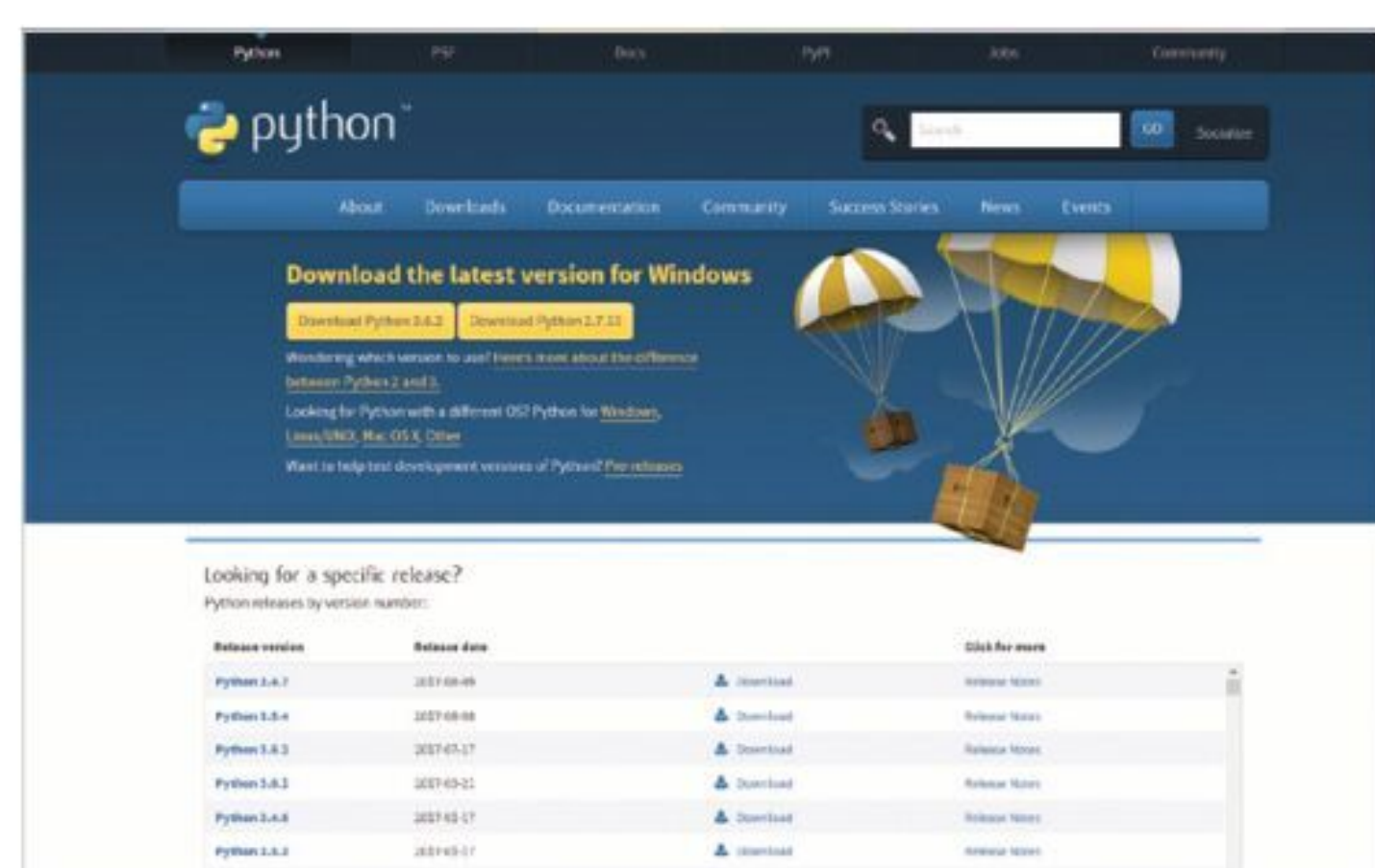
Whilst a text editor is an ideal environment to enter code into, it's not an absolute necessity. You can enter and execute code directly from the IDLE but a text editor, such as Sublime Text or Notepad++, offers more advanced features and colour coding when entering code.

☐ 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.

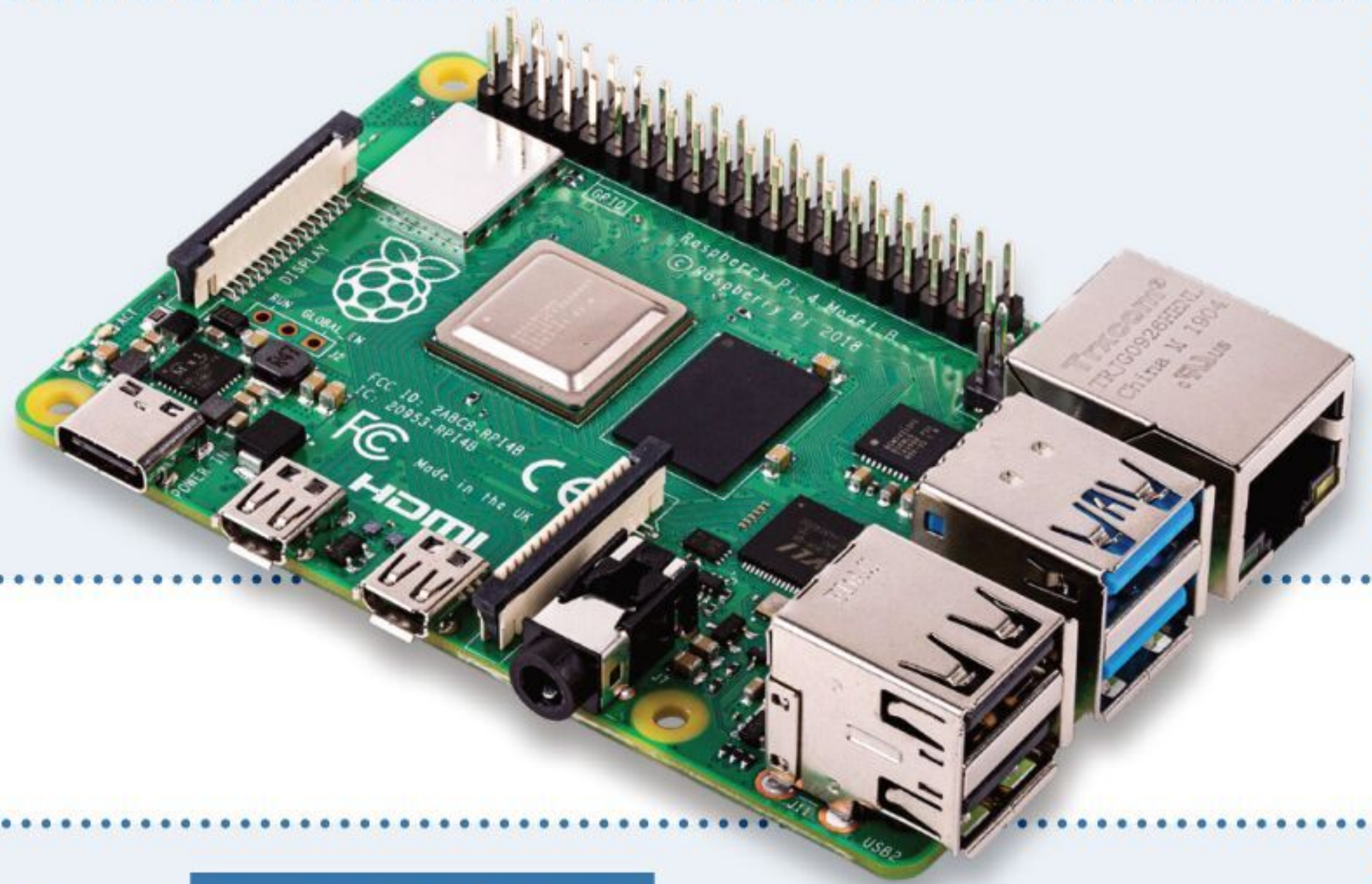


THE RASPBERRY PI

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 4 Model B is the latest version, incorporating a more powerful CPU, a choice of 1GB, 2GB or 4GB memory versions and Wi-Fi and Bluetooth support. You can pick up a Pi from around £34, increasing up to £54 for the 4GB memory version, or as a part of kit for £50+, depending on the kit you're interested in.



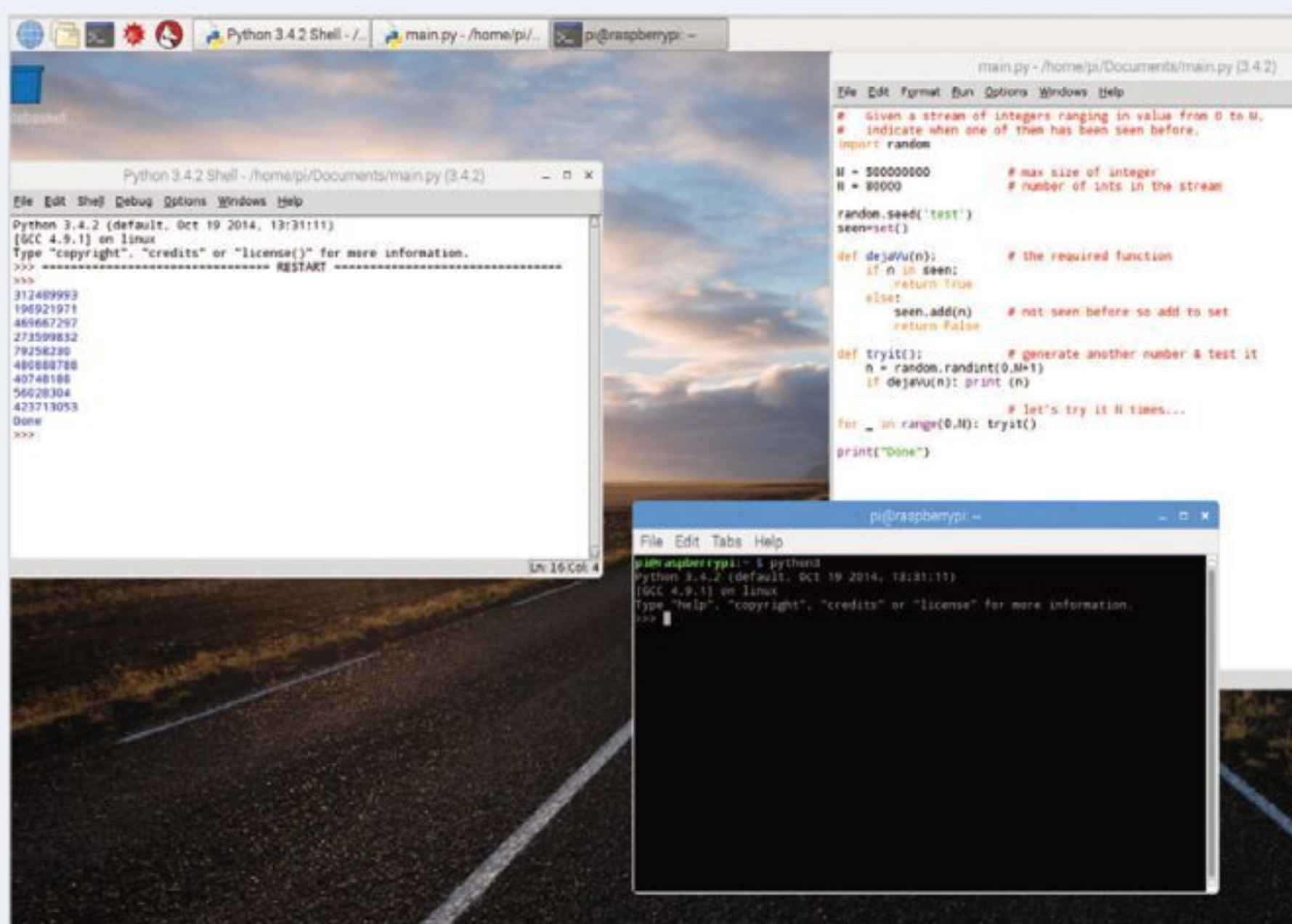
FUZE PROJECT

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.



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.



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.





Guido van Rossum, the designer of Python.

use Python to study large quantities of data; in April 2019 the first

Project), the combined power of the telescopes formed an

used to describe huge to around 5,000 years' worth of MP3 files. Once all those hard drives

[illegible]

Python code is both elegant and easy to read (when you know how).



Made up from over 5 Petabytes of data, spread across a ton of hard drives, Python helped science to unveil the first image of a black hole.

As a side note, it's not just the likes of the Stock Exchange that use Python to study large quantities of data; in April 2019 the first image of a black hole was released, the supermassive black hole in the galaxy called M87, located roughly 55 million light years away. Thanks to the collaboration of over 200 scientists, using an array of telescopes spanning the world (called the Event Horizon Telescope Project), the combined power of the telescopes formed an impressive five petabytes of data, spread across tens of hard drives weighing in at nearly one ton. Five petabytes, by the way, equates to around 5,000 years' worth of MP3 files. Once all those hard drives were gathered together and shipped to a central supercomputer cluster, the team then used Python to painstakingly stitch together all the fragments of data from the five petabytes to finally form the most talked about astronomic event of the decade.

AI, if you're not familiar with the term, stands for Artificial Intelligence. Although we're still a long way off from the visionary stories of Arthur C. Clarke, AI is fast becoming one of the most influential technologies of our modern age. Rather than controlling robots, the AI that Python drives is designed to create autonomous ways of interacting with people online. For example, when you search for something on the Amazon website you will usually notice that similar products start to appear, whether within Amazon itself,

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.



Python 3 is the best option to download, or update to.

Python 3 is the newest release of the programming language. However, if you dig a little deeper into the Python site and investigate Python code online, you will undoubtedly come across Python 2. Crucially, although you can run Python 3 and Python 2 alongside each other, it's not recommended. Always opt for the latest stable release, as posted by the Python website. You will find, when using macOS or Linux, that Python 2 is already installed. This is because both these operating systems utilise elements of code necessary to the core functionality of the OS. Linux users tend to be better off, as most distributions package the latest version of Python 3 out-of-the-box, whereas macOS often has Python 3, it's usually an older version. Microsoft doesn't use any Python code for its core Windows systems, which is why you won't find Python inherent to Windows and therefore need to install it from scratch.

You need to be careful when you look up Python code online, although there are countless websites that offer quick tutorials, code snippets and support – and 99% of these sites are a great help to those starting out with Python – a lot of the sites haven't been updated for some time, and as such use Python 2. If you enter Python 2 code into the Python 3 IDLE, the chances are it won't work due to incompatibilities between the older version and the newer. Python 2 is good, but Python 3 is better. You can obviously spend time converting the Python 2 code into version 3, but, to begin with, it's best to make sure that the code you're looking at is for the Python 3 libraries. Don't worry, though, all the code in this book is designed for Python 3, and that includes all the sub versions from Python 3.1 to the latest 3.x.

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.

However, support for Python 2 is set to end on January 1st 2020, but this doesn't mean it'll be the last you see of it. Many Linux distros use Python 2 libraries, as does macOS, and to be fair, for the developers to transfer the existing Python 2 content to Python 3 may take more time than they have available, i.e. before the start of 2020. It's likely then, that we will still be seeing Python 2 long after it has had the final nail hammered into its coffin – in fact, expect to see that cut-off date extend further into the future.

or from a search engine, or even Facebook. The code behind these targeted snippets is Python, and it's using a form of AI to help determine what it is you would likely search for.

Despite the fact that many people find the targeting of advertising intrusive, or even an invasion of privacy, you have to admit that the code technology behind it all is rather impressive. With some very clever techniques, a Python developer is able to create a machine thinking script that not only displays items, news stories, books, other websites and ideas relating to what you've searched for, but it can also predict what you may be interested in looking for in the future. Another element to consider, with regards to AI, is that Python code can be used to help a computer learn how to do something more efficiently. In the case of neural networks in AI, the Python code is designed to return a result, then, as the code is run over and over again, the AI portion will begin to learn how to obtain a more accurate result, or do the maths behind the code quicker. It all depends on what the developer wants from their AI Python code.

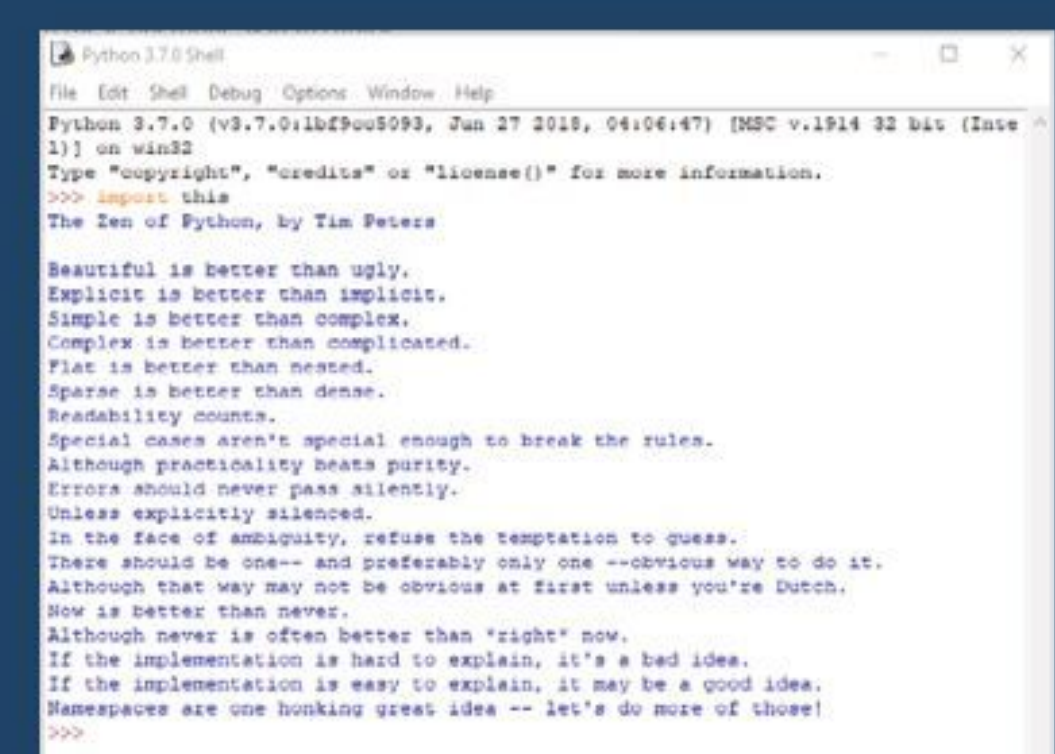


An example of Python AI code, using a feedforward neural network.

```
import random
import numpy as np
class Network(object):
    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes[1:])]
    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
    def SGD(self, training_data, epochs, mini_batch_size, eta,
            test_data=None):
```

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.



The Zen of Python, as seen when entering: `import this` into the Python IDLE.

As you will discover over the coming pages, Python is a fantastic language to learn. Get to grips with the basics, and before long, you'll be creating your own Python code for games, tools, and maybe even something in AI. The only limit with Python is your own imagination.



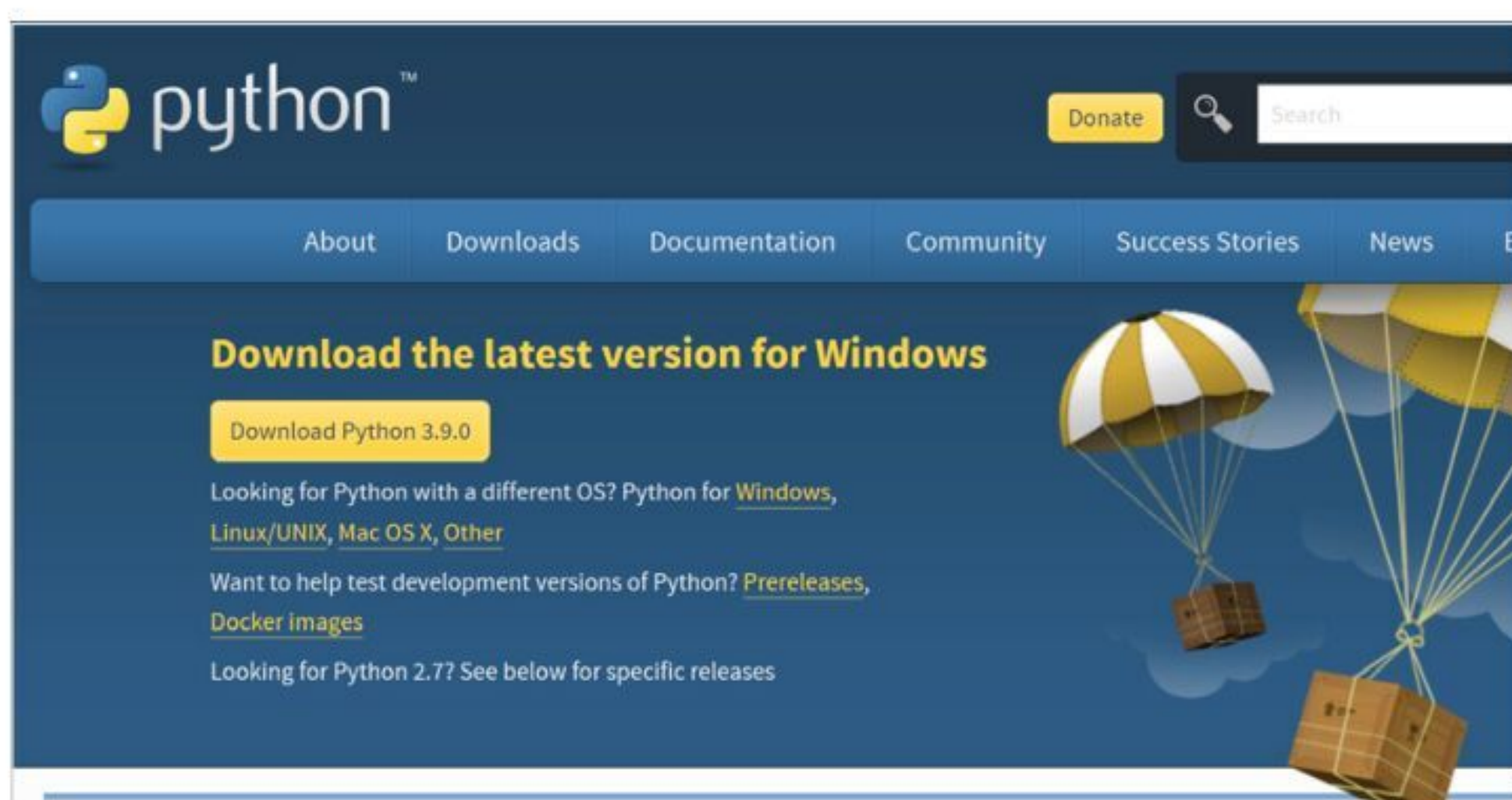
How to Set Up Python in Windows

Windows users can easily install the Python programming language via the main Python Downloads page. While 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.

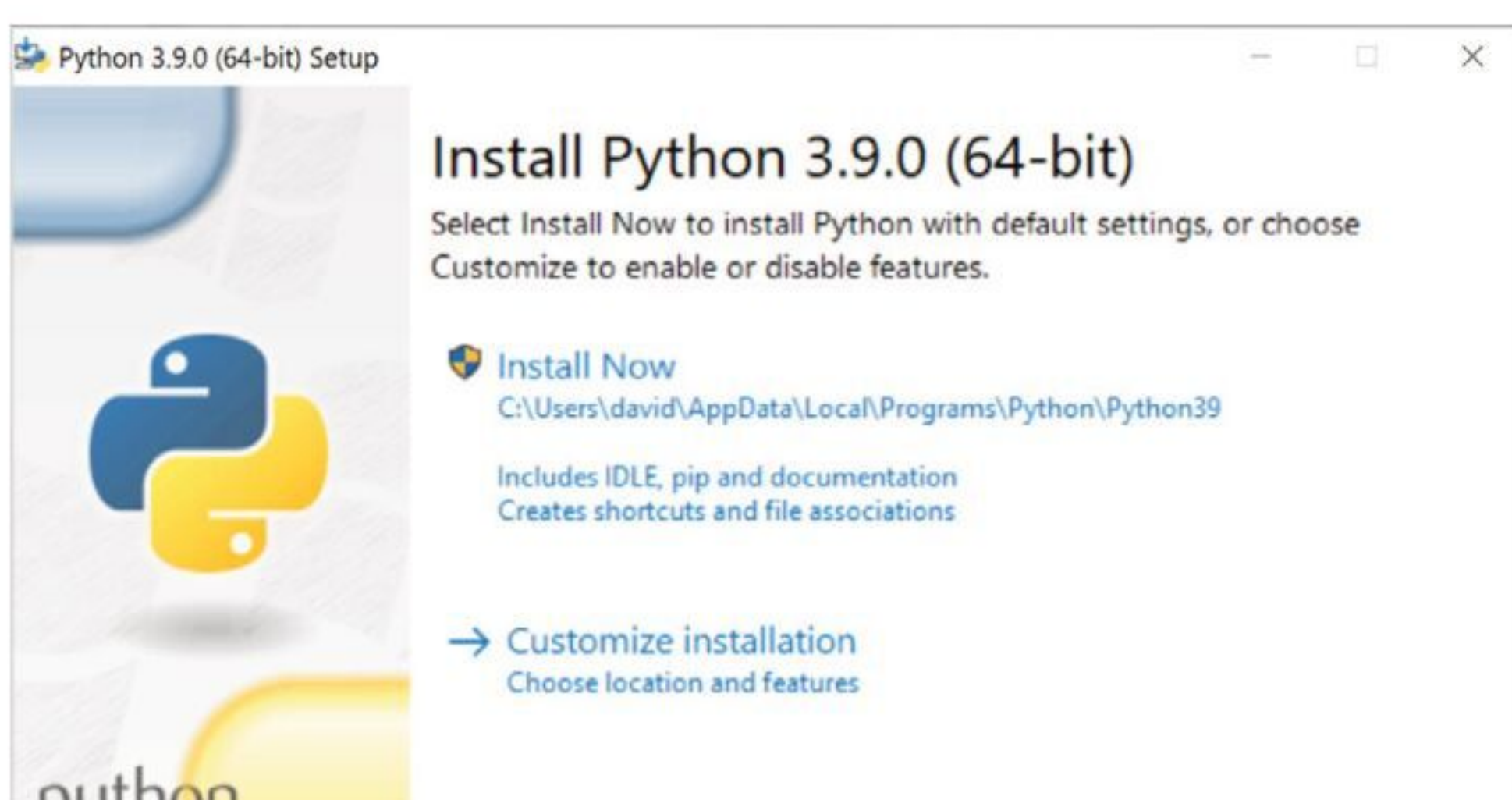
INSTALLING PYTHON 3.X

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

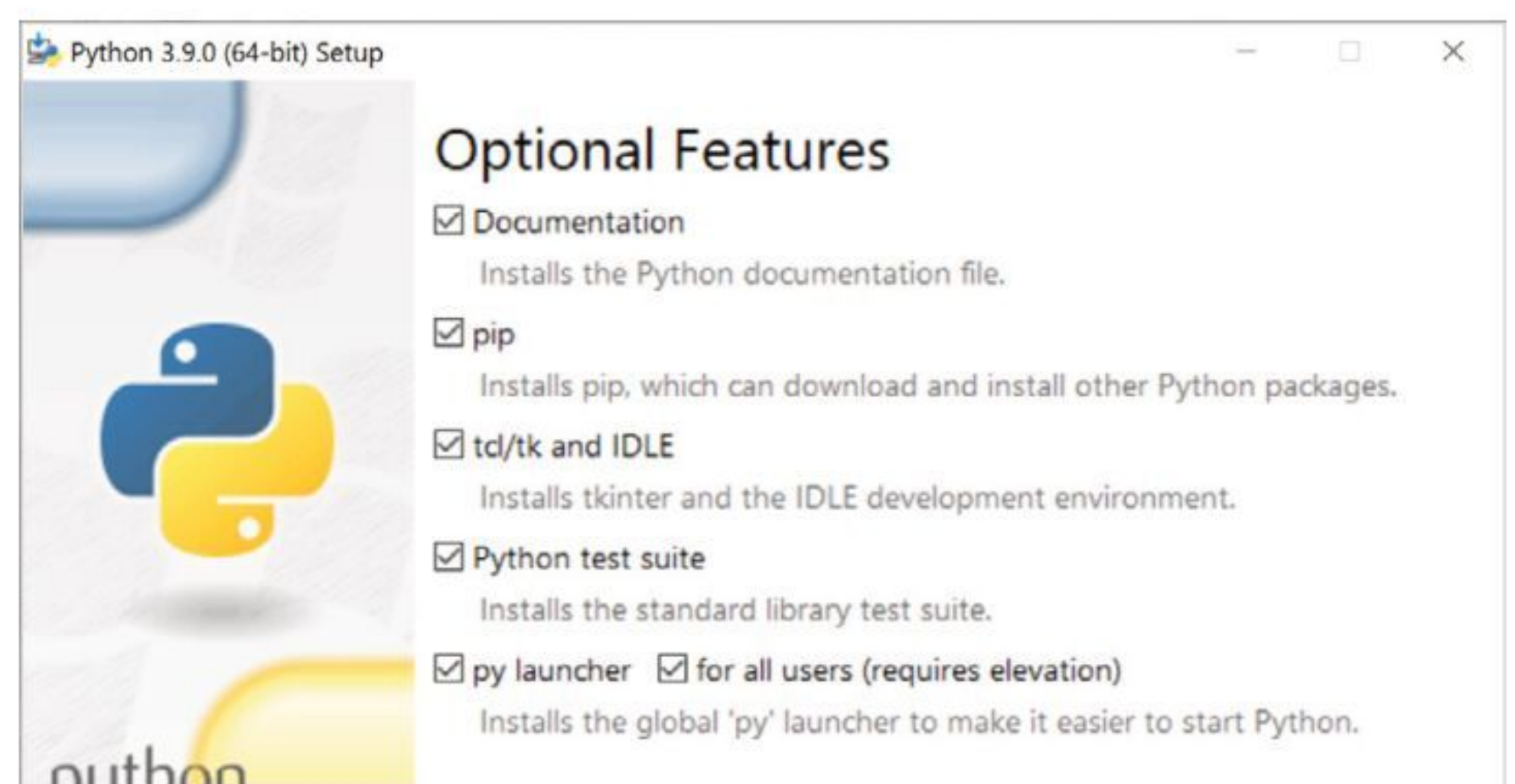
STEP 1 Start by opening your web browser to **www.python.org/downloads/**. Look for the button detailing the Download link for Python 3.x. Python is regularly updated, changing the last digit for each bug fix and update. Therefore, don't worry if you see Python 3.7.0, or more, as long as it's Python 3, the code in this book will work fine.



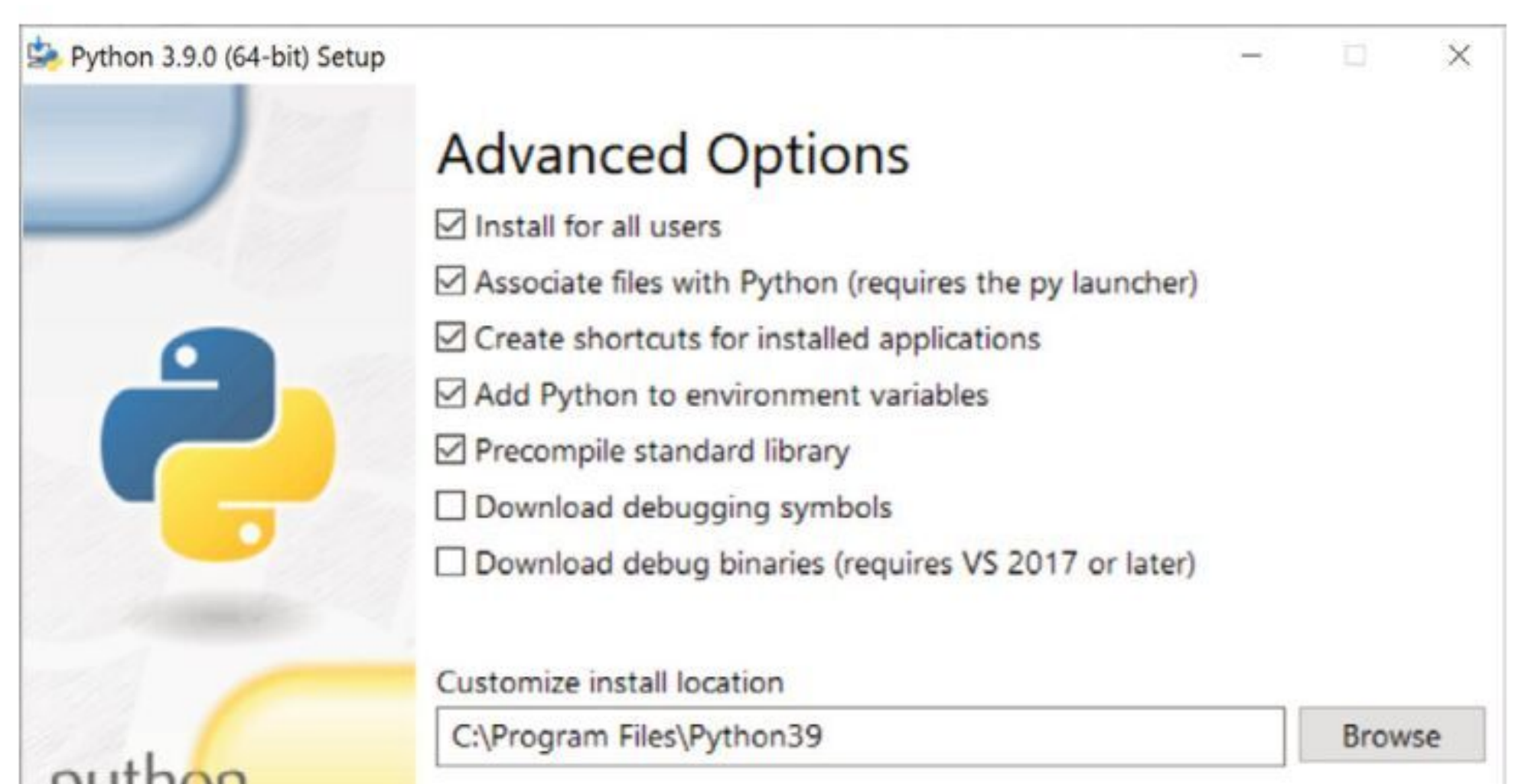
STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you'll have two choices: Install Now and Customise Installation. We recommend opting for the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



STEP 3 Choosing the Customise option allows you to specify certain parameters, and while 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.

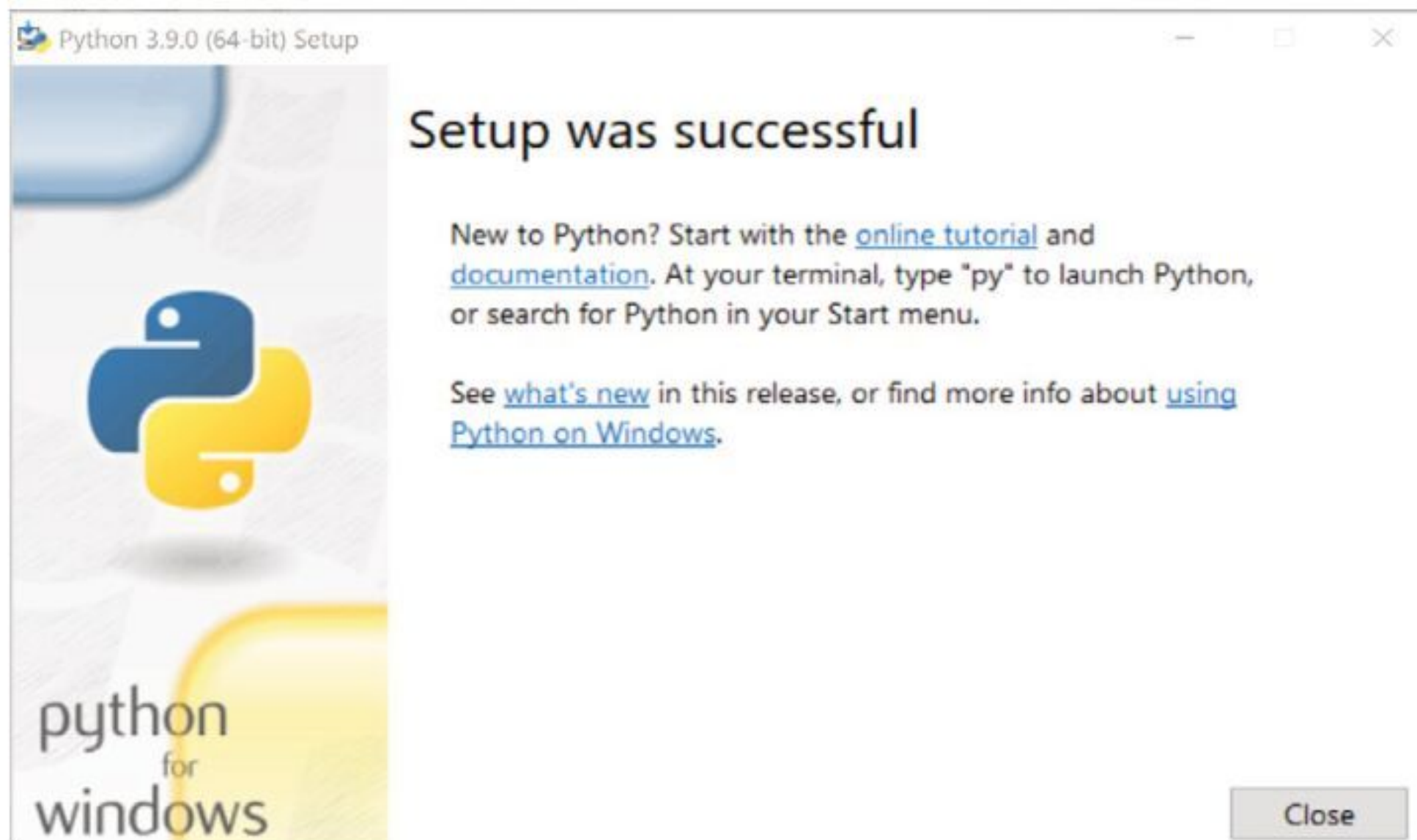


STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library, and Install for All Users options are ticked – these make using Python later much easier. Click Install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

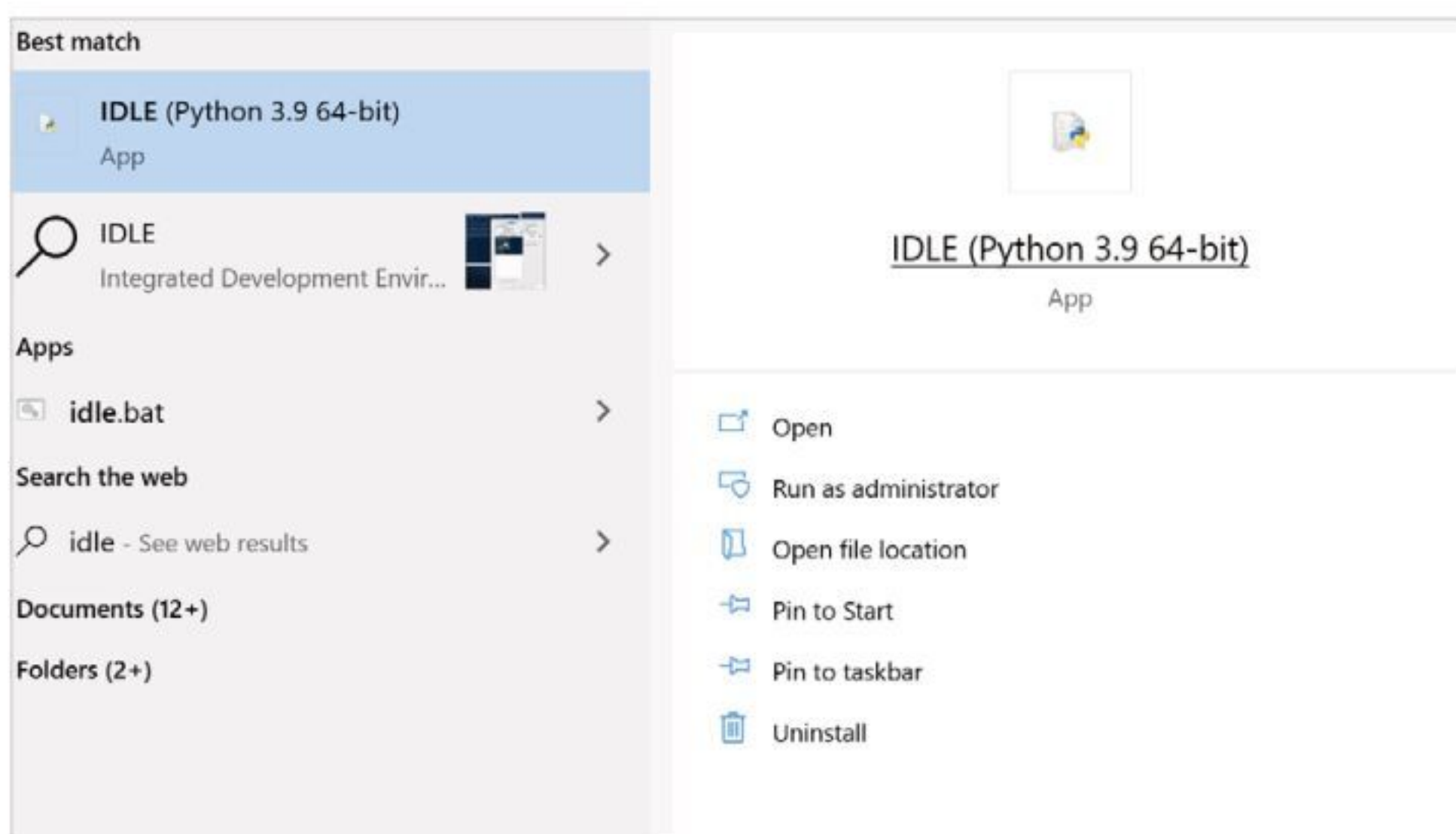


STEP 5

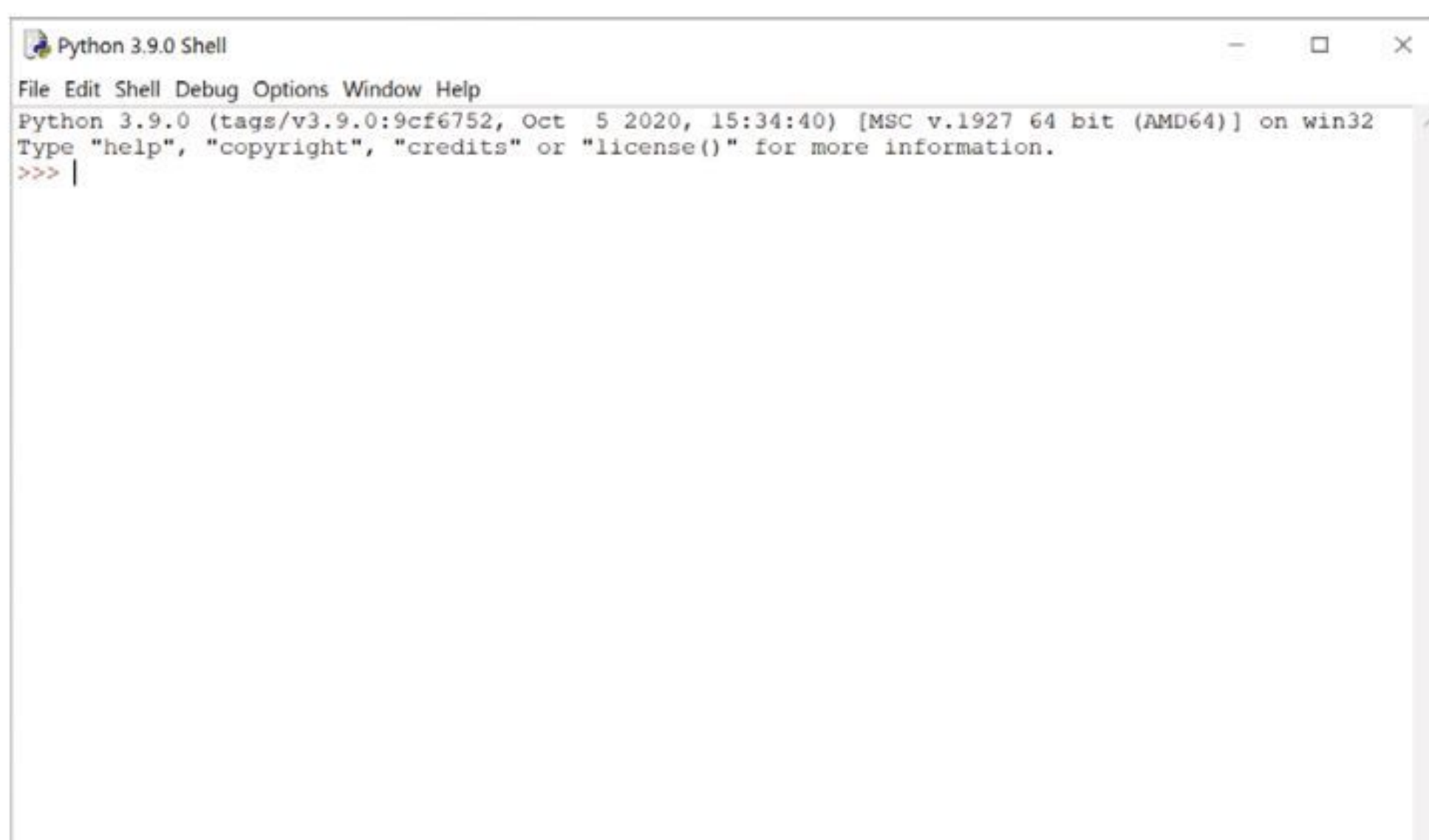
You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.

**STEP 6**

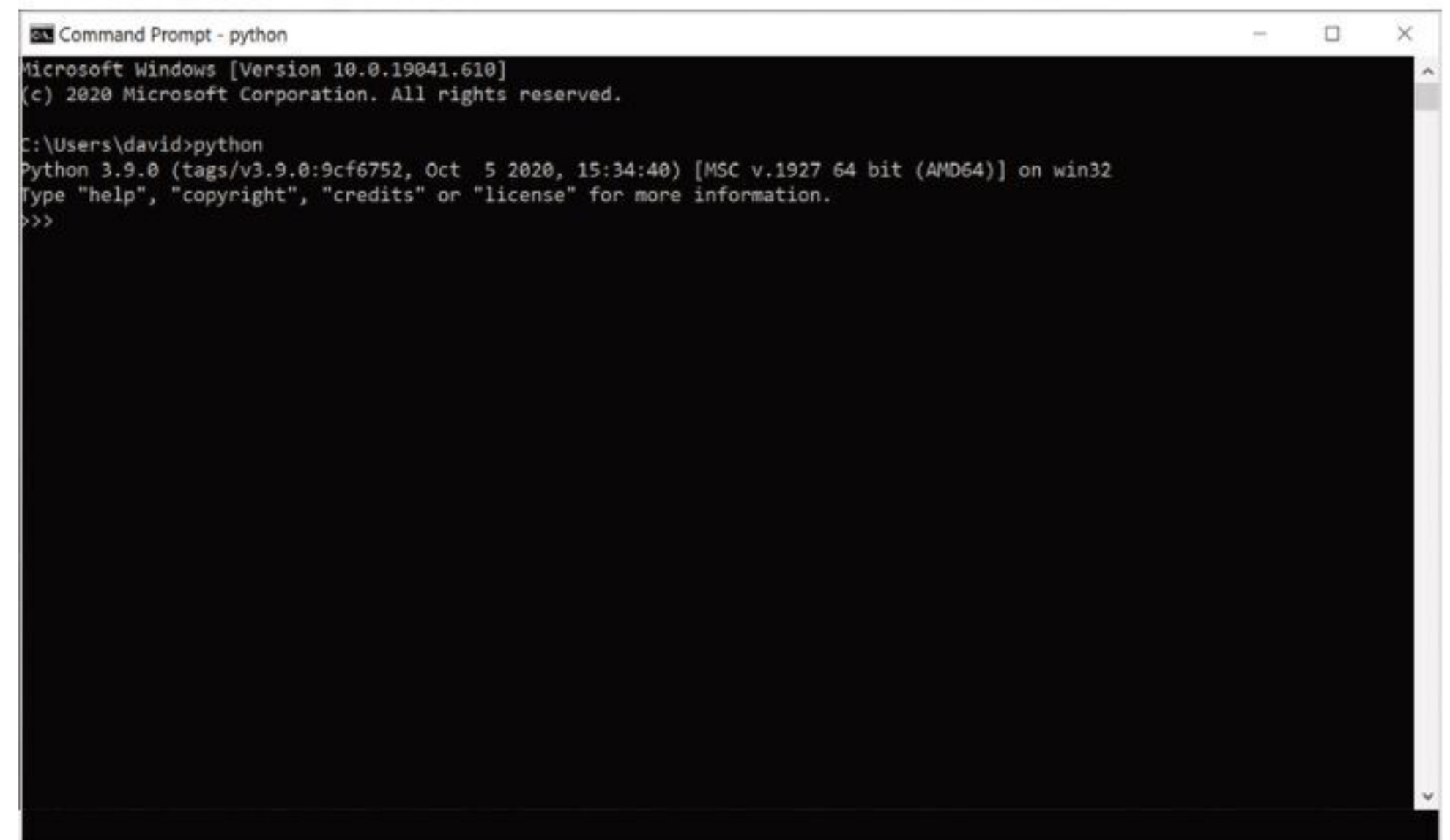
Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.x (64-bit) – or 32-Bit (more on that in a moment) - will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.

**STEP 7**

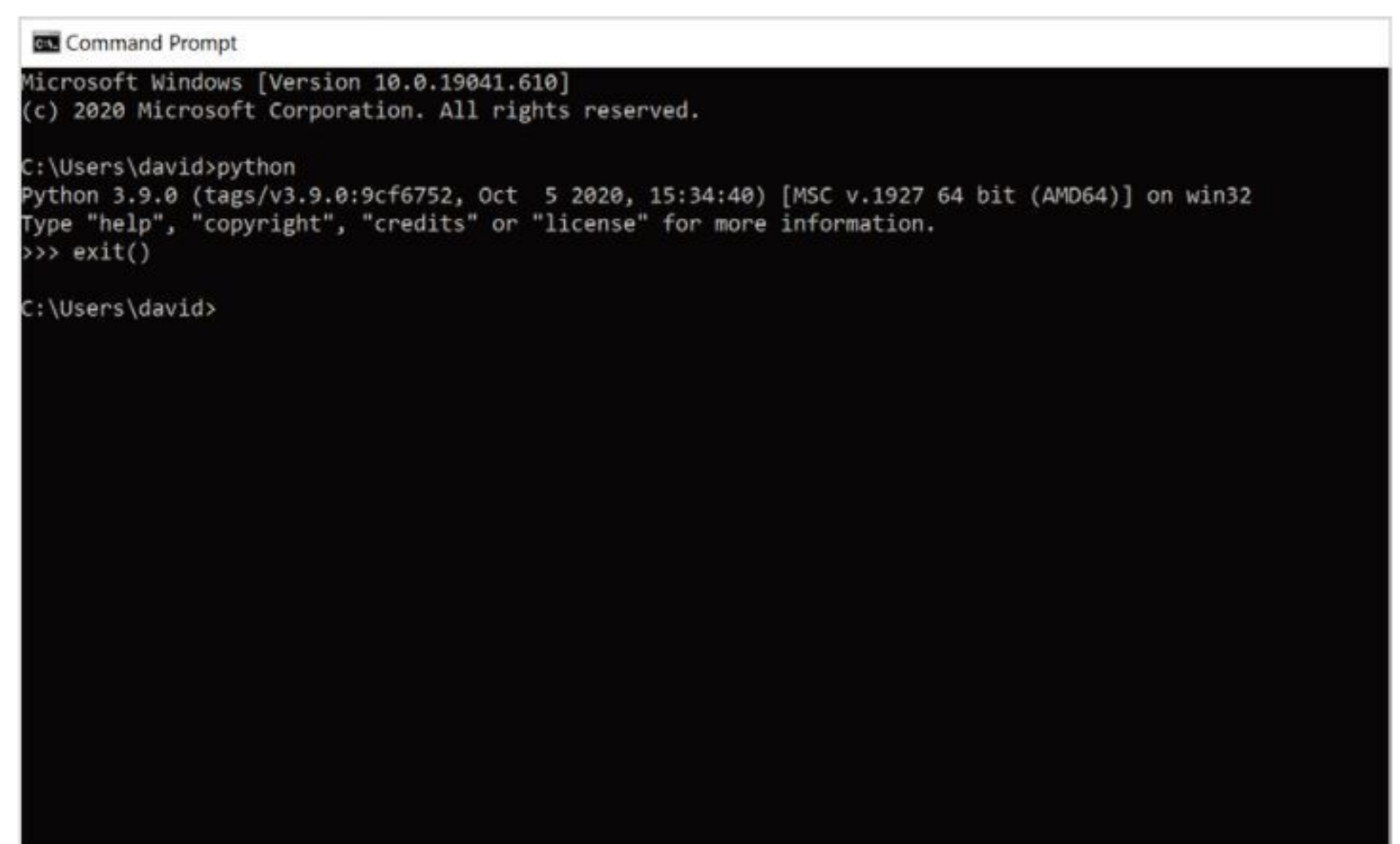
Clicking on the IDLE (Python 3.x 64-bit) link will launch the Python Shell, where we'll begin our Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.

**STEP 8**

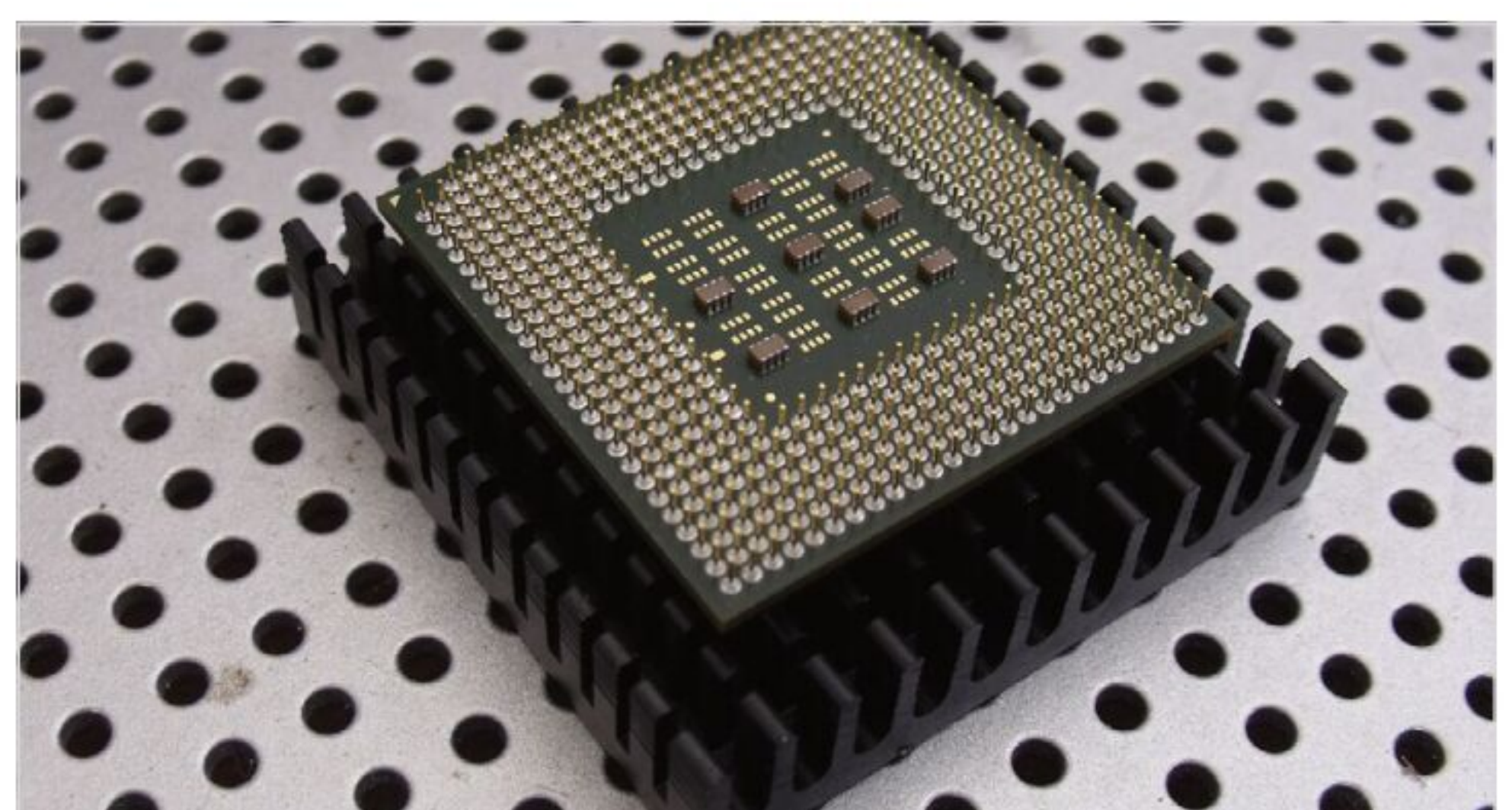
If you now click on the Windows Start button again, and this time type CMD you'll be presented with the Command Prompt link. Click it and you'll be at the Windows command line environment. To enter Python within the command line, you need to type: `python` and press Enter.

**STEP 9**

The command line version of Python works in much the same way as the Shell you opened in Step 7; note the three left-facing arrows (>>>). While it's a perfectly fine environment, it's not too user-friendly, so we'll leave the command line for now. Enter `exit()` to leave, and close the Command Prompt window.

**STEP 10**

There are two versions of the latest Python available, 32-Bit and 64-Bit. Essentially, the 64-Bit version of Python allows you to utilise more than 4GB of memory. While this is great for users with a lot of memory, and memory hungry processes, it's not an essential feature of Python. However, the 32-bit version may provide more compatibility with existing modules.





How to Set Up Python in Linux

While the Raspberry Pi's operating system contains the latest, stable version of Python, other Linux distros don't come with Python 3 pre-installed. If you're not going down the Pi route, then here's how to check and install Python for Linux.

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 for this particular tutorial we will stick to Linux Mint.

STEP 1

First you need to ascertain which version of Python is currently installed in your Linux system. To begin with, drop into a Terminal session from your distro's menu, or hit the Ctrl+Alt+T keys.

```
david@david-Mint: ~  
File Edit View Search Terminal Help  
david@david-Mint:~$
```

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. Most Linux distro come with both Python 2 and 3 by default, as there's plenty of code out there still available for Python 2. Now enter: `python3 --version`.

```
david@david-Mint: ~  
File Edit View Search Terminal Help  
david@david-Mint:~$ python --version  
Python 2.7.15rc1  
david@david-Mint:~$ python3 --version  
Python 3.6.7  
david@david-Mint:~$
```

STEP 3

In our case we have both Python 2 and 3 installed. As long as Python 3.x.x is installed, then the code in our tutorials will work. It's always worth checking to see if the distro has been updated with the latest versions, enter: `sudo apt-get update && sudo apt-get upgrade` to update the system.

```
david@david-Mint: ~  
File Edit View Search Terminal Help  
david@david-Mint:~$ python --version  
Python 2.7.15rc1  
david@david-Mint:~$ python3 --version  
Python 3.6.7  
david@david-Mint:~$ sudo apt-get update && sudo apt-get upgrade  
[sudo] password for david:
```

STEP 4

Once the update and upgrade completes, enter: `python3 --version` again to see if Python 3.x is updated, or even installed. As long as you have Python 3.x, you're running the most recent major version, the numbers after the 3. indicate patches plus further updates. Often they're unnecessary, but they can contain vital new elements.

```
File Edit View Search Terminal Help  
Need to get 1,409 kB of archives.  
After this operation, 23.6 kB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libasound2 amd64 1.1.3-5ubuntu0.2 [359 kB]  
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libasound2-data all 1.1.3-5ubuntu0.2 [36.5 kB]  
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 linux-libc-dev amd64 4.15.0-44.47 [1,013 kB]  
Fetched 1,409 kB in 0s (3,023 kB/s)  
(Reading database ... 290768 files and directories currently installed.)  
Preparing to unpack .../libasound2 1.1.3-5ubuntu0.2 amd64.deb ...  
Unpacking libasound2:amd64 (1.1.3-5ubuntu0.2) over (1.1.3-5ubuntu0.1) ...  
Preparing to unpack .../libasound2-data 1.1.3-5ubuntu0.2 all.deb ...  
Unpacking libasound2-data (1.1.3-5ubuntu0.2) over (1.1.3-5ubuntu0.1) ...  
Preparing to unpack .../linux-libc-dev 4.15.0-44.47 amd64.deb ...  
Unpacking linux-libc-dev:amd64 (4.15.0-44.47) over (4.15.0-43.46) ...  
Setting up libasound2-data (1.1.3-5ubuntu0.2) ...  
Setting up linux-libc-dev:amd64 (4.15.0-44.47) ...
```

STEP 5

However, if you want the latest, cutting edge version, you'll need to build Python from source. Start by entering these commands into the Terminal:

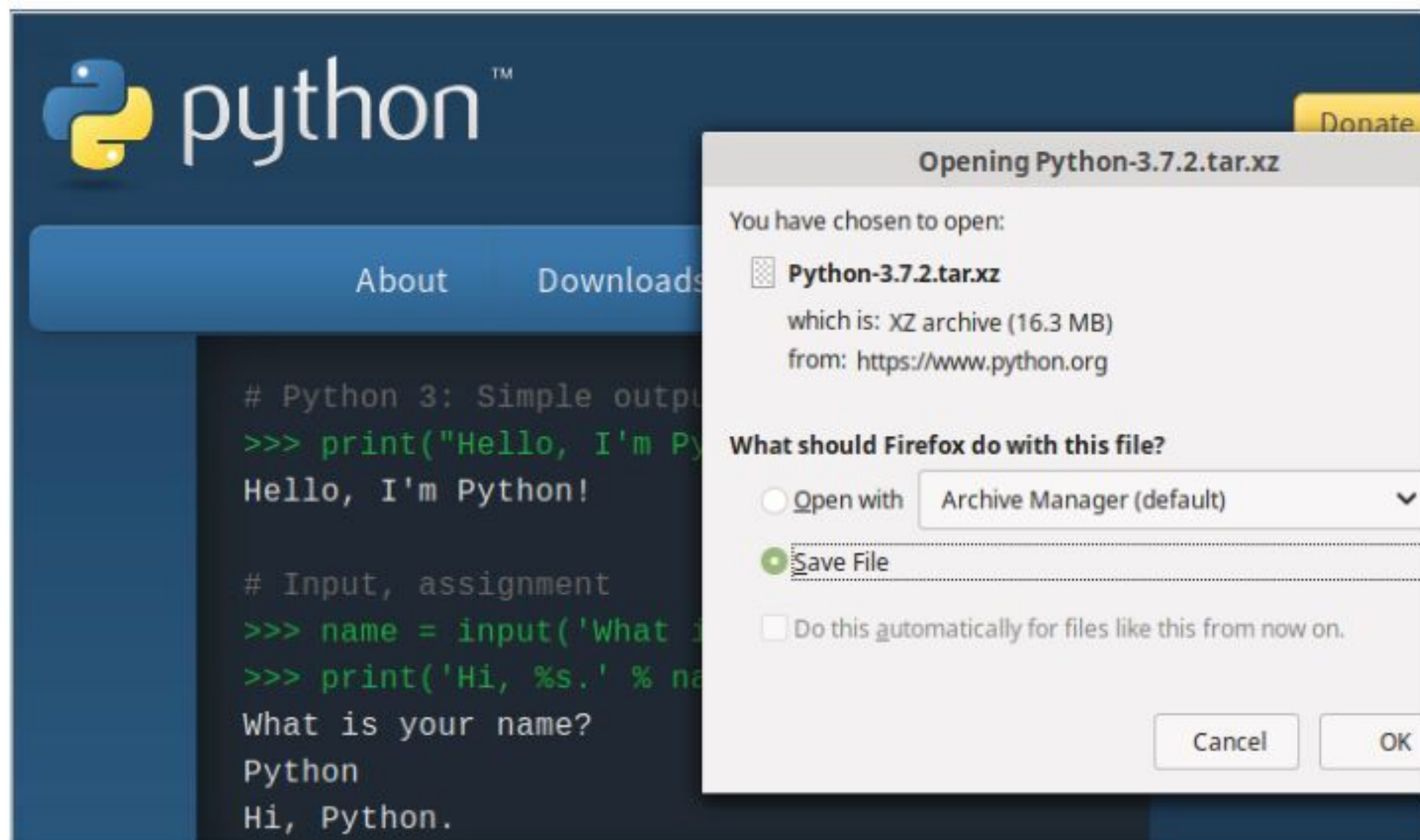
```
sudo apt-get install build-essential checkinstall  
sudo apt-get install libreadline-gplv2-dev  
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev  
libgdbm-dev libc6-dev libbz2-dev
```

```
david@david-Mint: ~  
File Edit View Search Terminal Help  
david@david-Mint:~$ sudo apt-get install build-essential checkinstall  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
build-essential is already the newest version (12.4ubuntu1).  
The following NEW packages will be installed  
checkinstall  
0 to upgrade, 1 to newly install, 0 to remove and 3 not to upgrade.  
Need to get 97.1 kB of archives.  
After this operation, 438 kB of additional disk space will be used.  
Do you want to continue? [Y/n] y
```



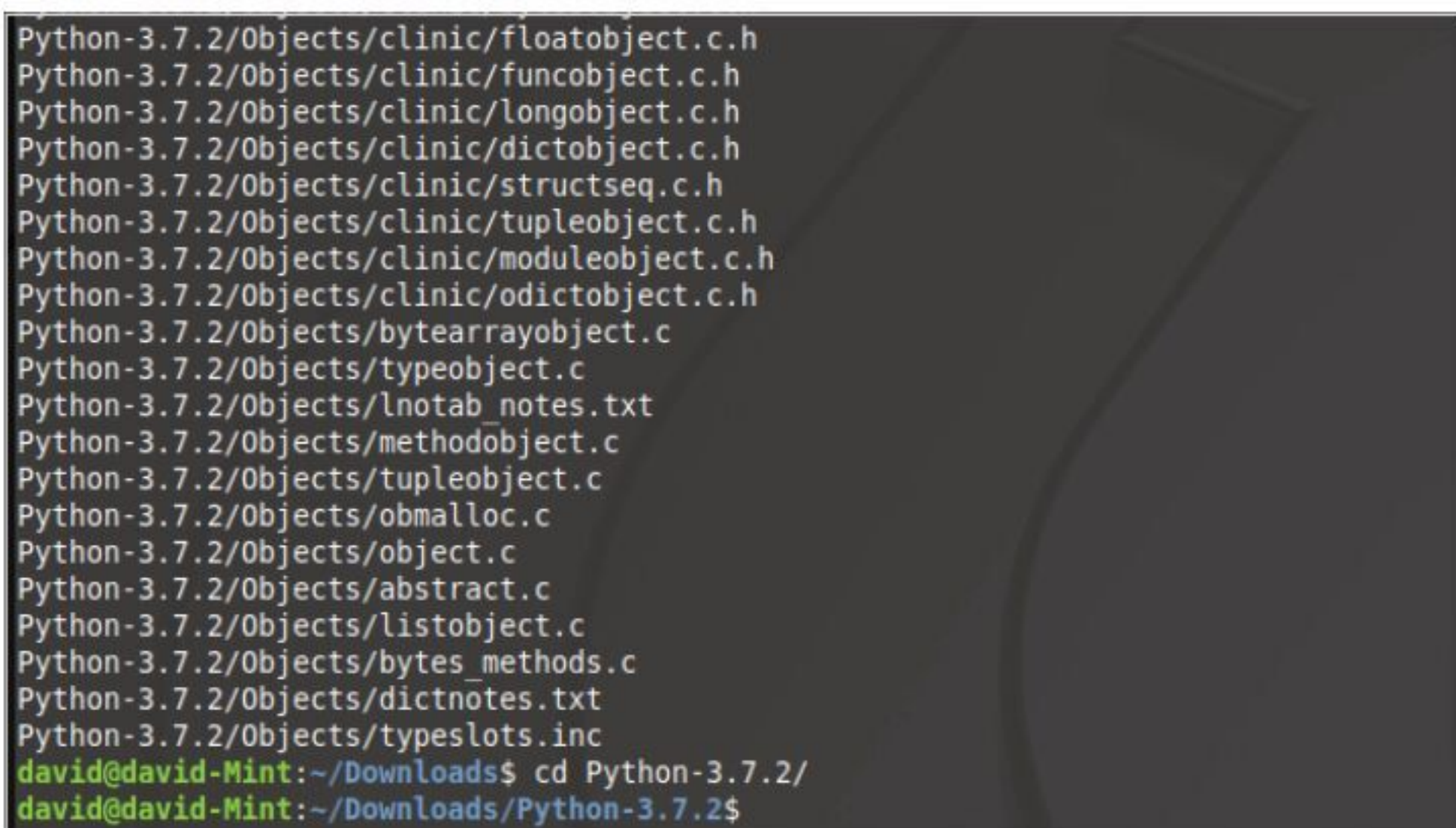

STEP 6

Open up your Linux web browser and go to the Python download page: <https://www.python.org/downloads>. Click on the Downloads, followed by the button under the Python Source window. This opens a download dialogue box, choose a location, then start the download process.



STEP 7

In the Terminal, go to the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.Y.Y.tar.xz` (replace the Y's with the version numbers you've downloaded). Now enter the newly unzipped folder with: `cd Python-3.Y.Y/`.

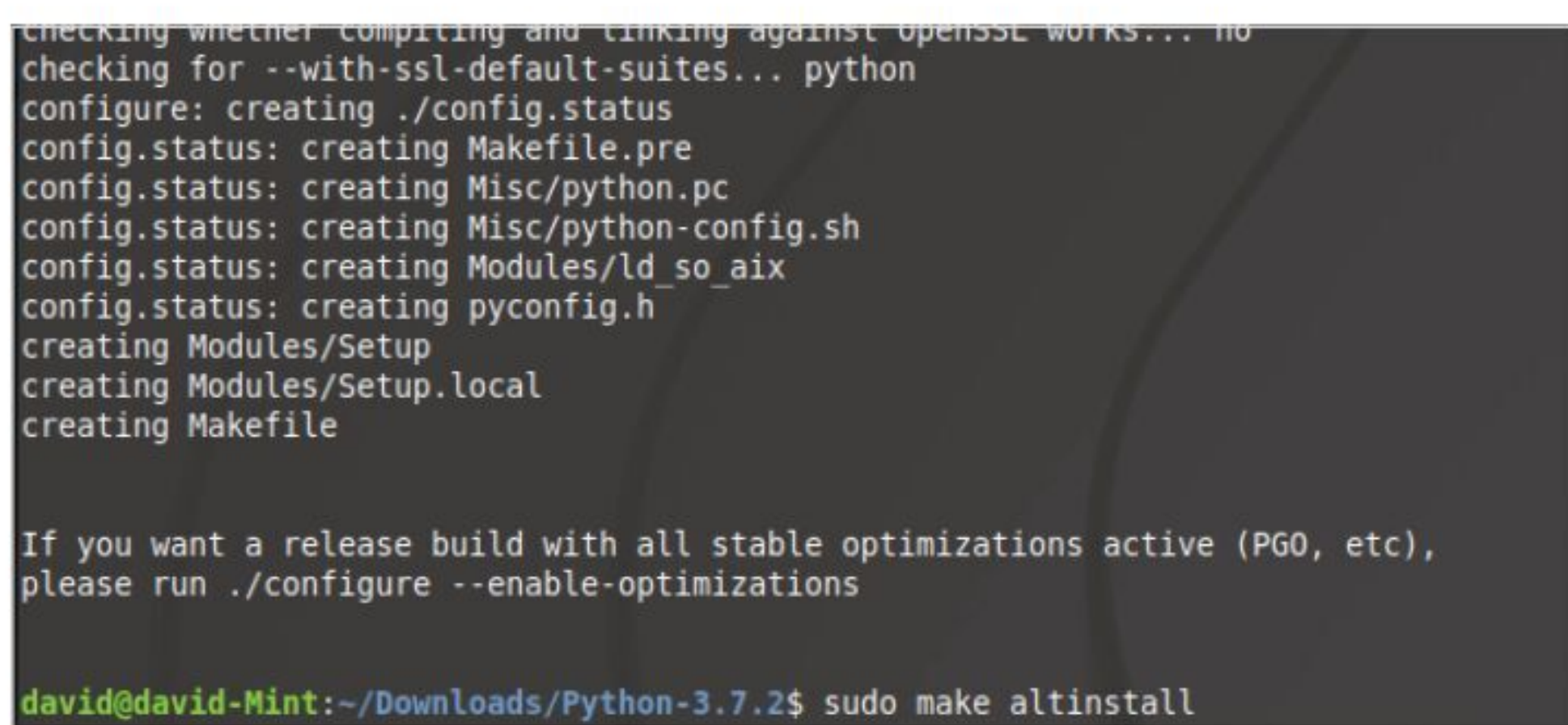


STEP 8

Within the Python folder, enter:

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

This could take a while, depending on the speed of your computer. Once finished, enter: `python3.7 --version` to check the latest installed version. You now have Python 3.7 installed, alongside older Python 3.x.x and Python 2.

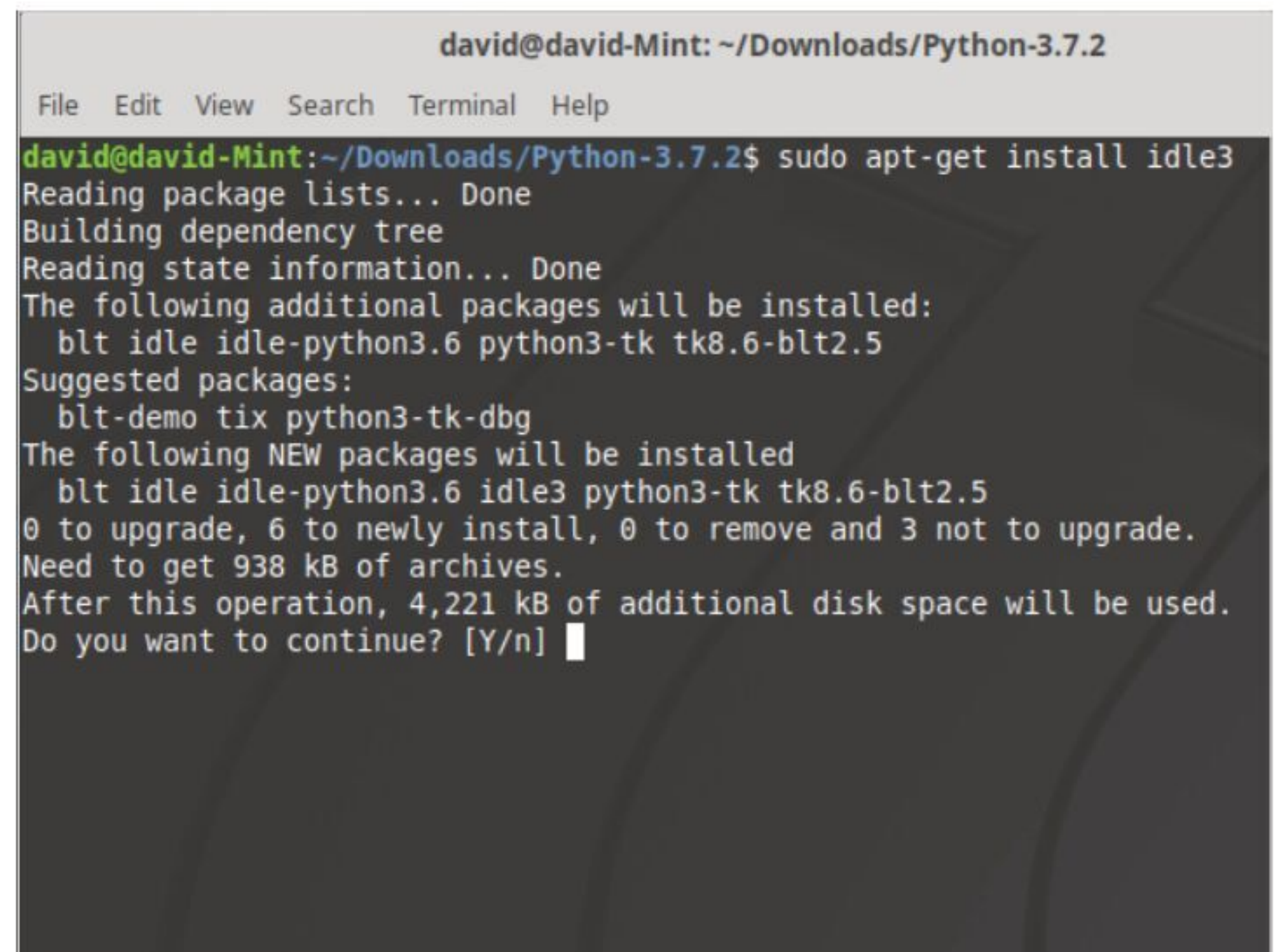


STEP 9

For the GUI IDLE, you'll 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 to the one you installed from source.



STEP 10

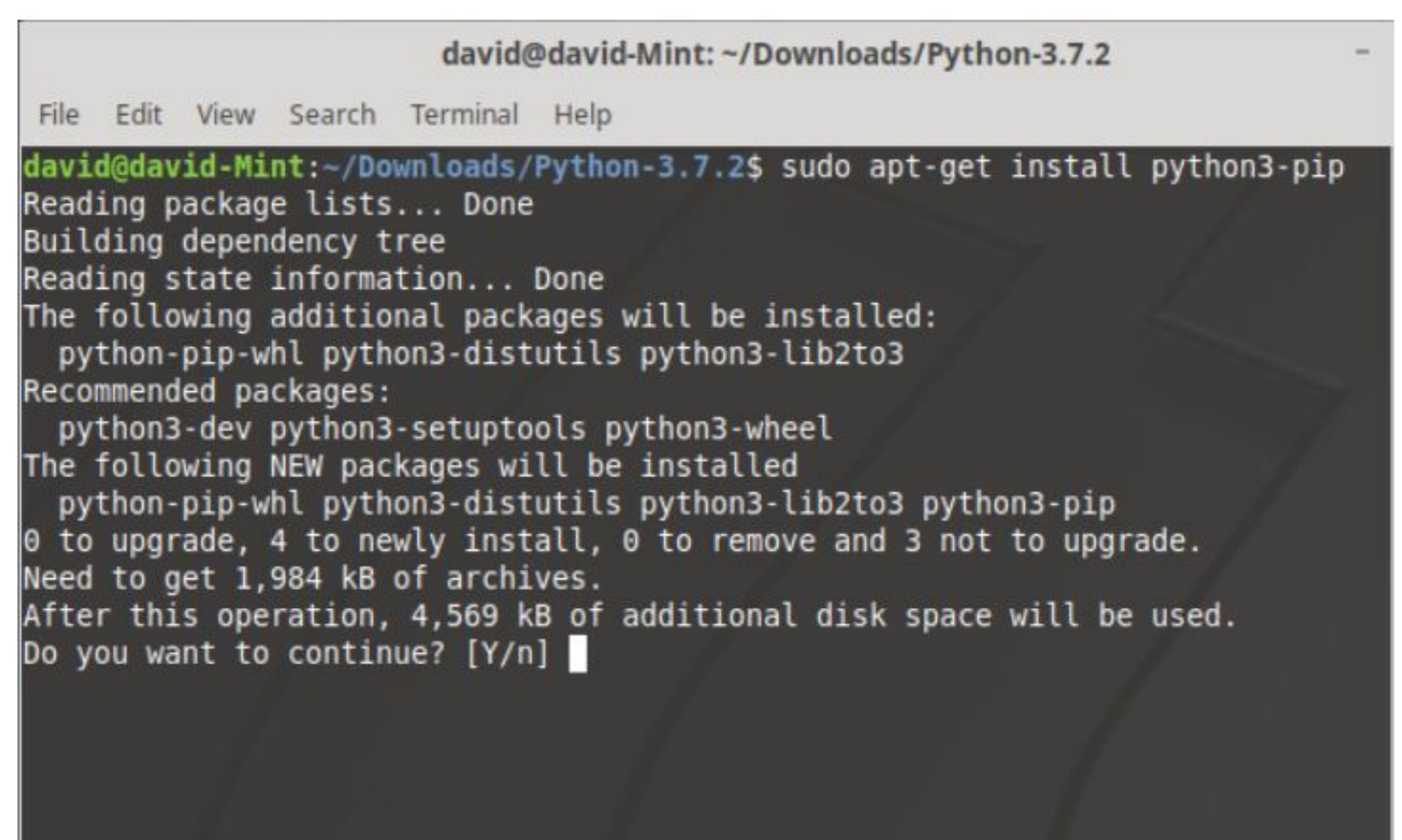
You'll also need PIP (Pip Installs Packages), which is a tool to help you install more modules and extras.

Enter: `sudo apt-get install python3-pip`

Once PIP is 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.



PYTHON ON macOS

Installation of Python on macOS can be done in much the same way as the Windows installation. Simply go to the Python webpage, hover your mouse pointer over the Downloads link and select Mac OS X from the options. You will then be guided to the Python releases for Mac versions, along with the necessary installers for macOS 64-bit for OS X 10.9 and later.



Python on the Pi

If you're considering on which platform to install and use Python, then give some thought to one of the best coding bases available: the Raspberry Pi. The Pi has many advantages for the coder: it's cheap, easy to use, and extraordinarily flexible.

THE POWER OF PI

While having a far more powerful coding platform on which to write and test your code is ideal, it's not often feasible. Most of us are unable to jump into a several hundred-pound investment when we're starting off and this is where the Raspberry Pi can help out.

While having a far more powerful coding platform on which to write and test your code is ideal, it's not often feasible. Most of us are unable to jump into a several hundred-pound investment when we're starting off and this is where the Raspberry Pi can help out.

The Raspberry Pi is a fantastic piece of modern hardware that has created, or rather re-created, the fascination we once all had about computers, how they work, how to code and foundation level electronics. Thanks to its unique mix of hardware and custom software, it has proved itself to be an amazing platform on which to learn how to code; specifically, using Python.

While you're able, with ease, to use the Raspberry Pi to learn to code with other programming languages, it's Python that has been firmly pushed to the forefront. The Raspberry Pi uses Raspbian as its recommended, default operating system. Raspbian is a Linux OS, or to be more accurate, it's a Debian-based distribution of Linux. This means that there's already a built-in element of Python programming, as opposed to a fresh installation of Windows 10, which has no Python-specific base. However, the Raspberry Pi Foundation has gone the extra mile to include a vast range of Python modules, extensions and even examples, out of the box. So, essentially, all you need to do is buy a Raspberry Pi, follow the instructions on how to set one up (by using one of our excellent Raspberry Pi guides found at www.bdmpublications.com) and you can start coding with Python as soon as the desktop has loaded.

Significantly, there's a lot more to the Raspberry Pi, which makes it an excellent choice for someone who is starting to learn how to code in Python. The Pi is remarkably easy to set up as a headless node. This means that, with a few tweaks here and there, you're able to remotely connect to the Raspberry Pi from any other computer, or device, on your home network. For example, once you've set up the remote connectivity options, you can simply plug the Pi into the power socket anywhere in your house within range of your wireless router. As long as the Pi is connected, you will be able to remotely access the desktop from Windows or macOS as easily as if you were sitting in front of the Pi with a keyboard and mouse.

Using this method saves a lot of money, as you don't need another keyboard, mouse and monitor, plus, you won't need to allocate sufficient space to accommodate those extras either. If you're pushed for space and money, then for around £60, buying one of the many

kits available will provide the Pi with a pre-loaded SD card (with the latest Raspbian OS), a case, power socket and cables, this is a good idea as you could, with very little effort, leave the Pi plugged into the wall under a desk, while still being able to connect to it and code.

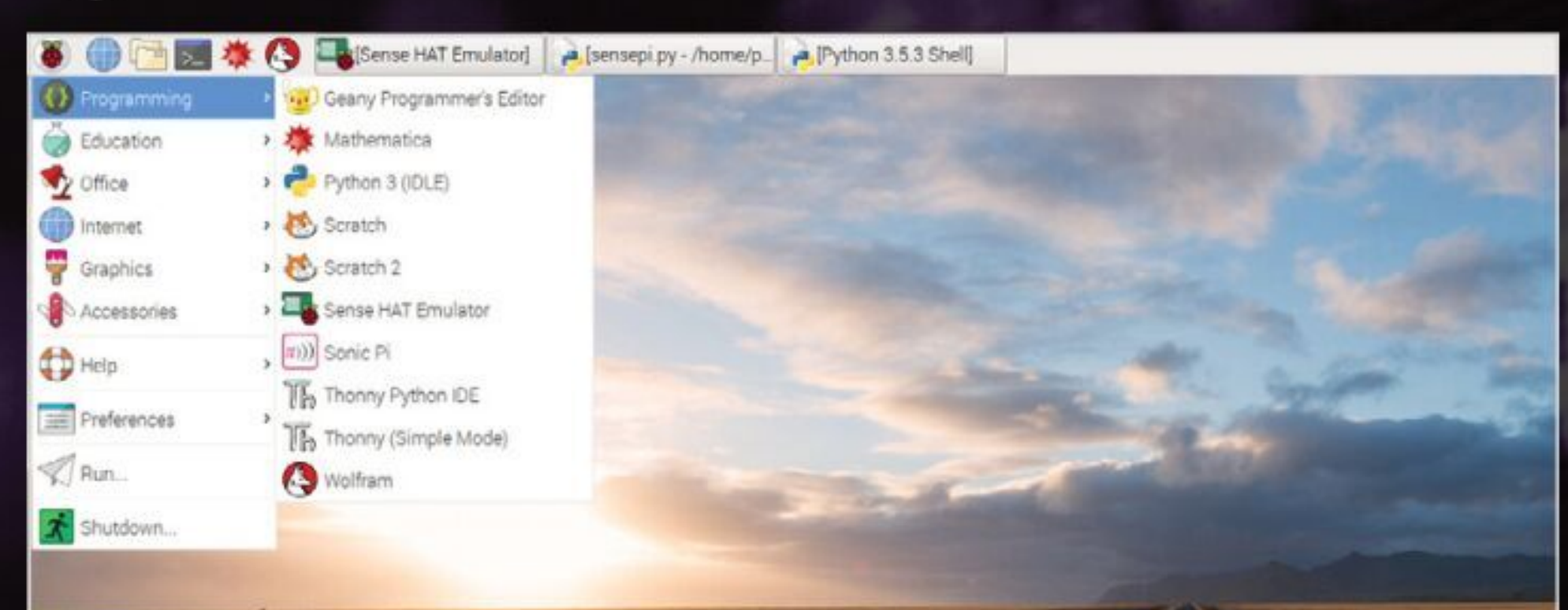
The main advantage is, of course, the extra content that the Raspberry Pi Foundation has included out of the box. The reason for this is that the Raspberry Pi's goal is to help educate the user, whether that's coding, electronics, or some other aspect of computing. To achieve that goal the Pi Foundation includes different IDEs for the user to compile Python code on; as well as both Python 2 and Python 3, there's even a Python library that allows you to communicate with Minecraft.

There are other advantages, such as being able to combine Python code with Scratch (an Object-Oriented programming language developed by MIT, for children to understand how coding works) and being able to code the GPIO connection on the Pi to further control any attached robotics or electronics projects. Raspbian also includes a Sense HAT Emulator (a HAT is a hardware attached piece of circuitry that offers different electronics, robotics and motorisation projects to the Pi), which can be accessed via Python code.

Consequently, the Raspberry Pi is an excellent coding base, as well as a superb project foundation. It is for these, and many other, reasons we've used the Raspberry Pi as our main Python codebase throughout this title. While the code is written and performed on a Pi, you're also able to use it in Windows, other versions of Linux and macOS. If the code requires a specific operating system, then, don't worry; we will let you know in the text.

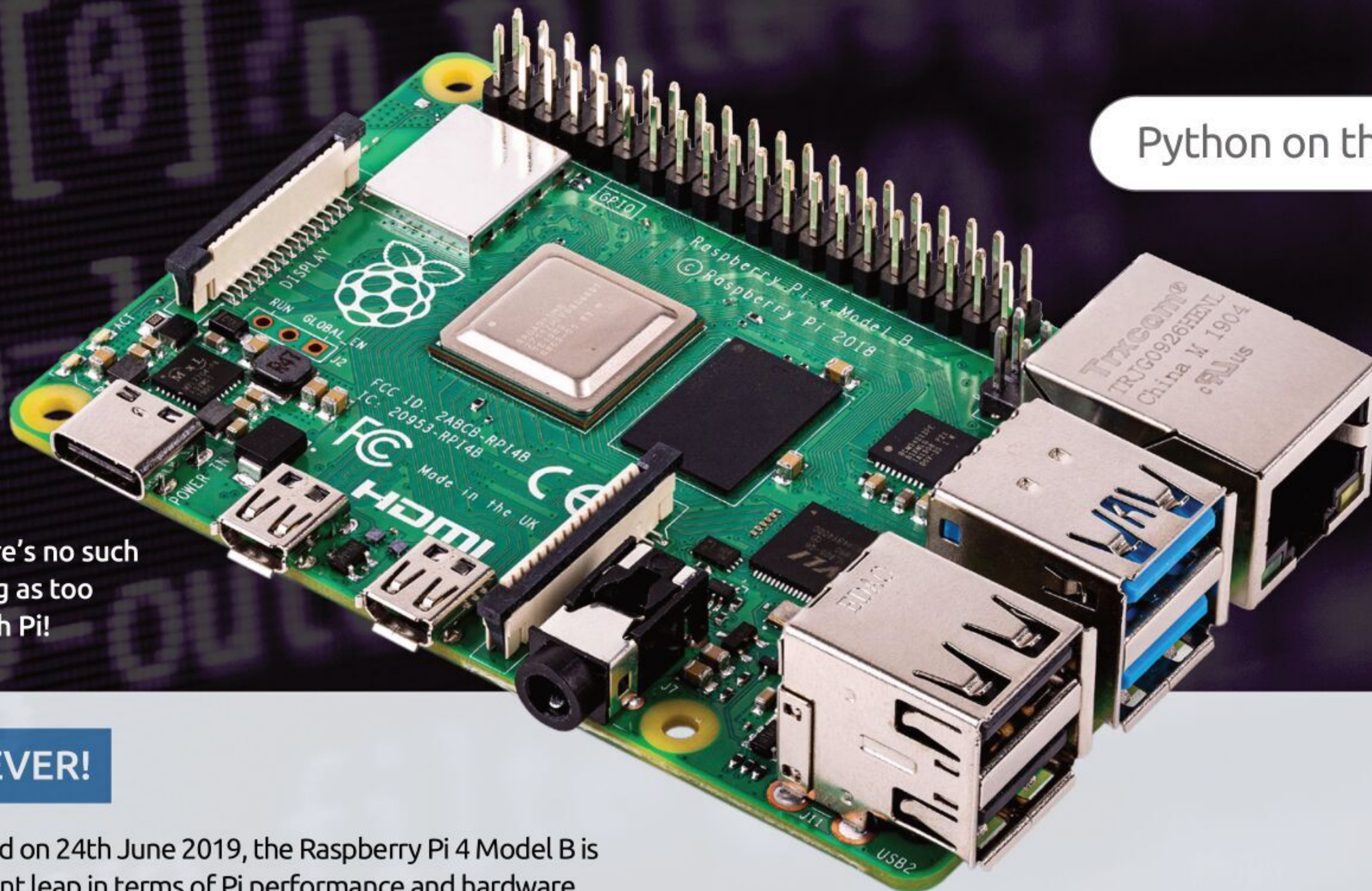


Everything you need to learn to code with Python is included with the OS!





There's no such thing as too much Pi!



PI 4-EVER!

Introduced on 24th June 2019, the Raspberry Pi 4 Model B is a significant leap in terms of Pi performance and hardware specifications. It was also one of the quickest models, aside from the original Pi, to sell out.

With a new 1.5GHz, 64-bit, quad-core ARM Cortex-A72 processor, and a choice of 2GB, 4GB, or 8GB memory versions, the Pi 4 is onestep closer to becoming a true desktop computer. In addition, the Pi 4 was launched with the startling decision to include dual-monitor support, in the form of a pair of two micro-HDMI ports.

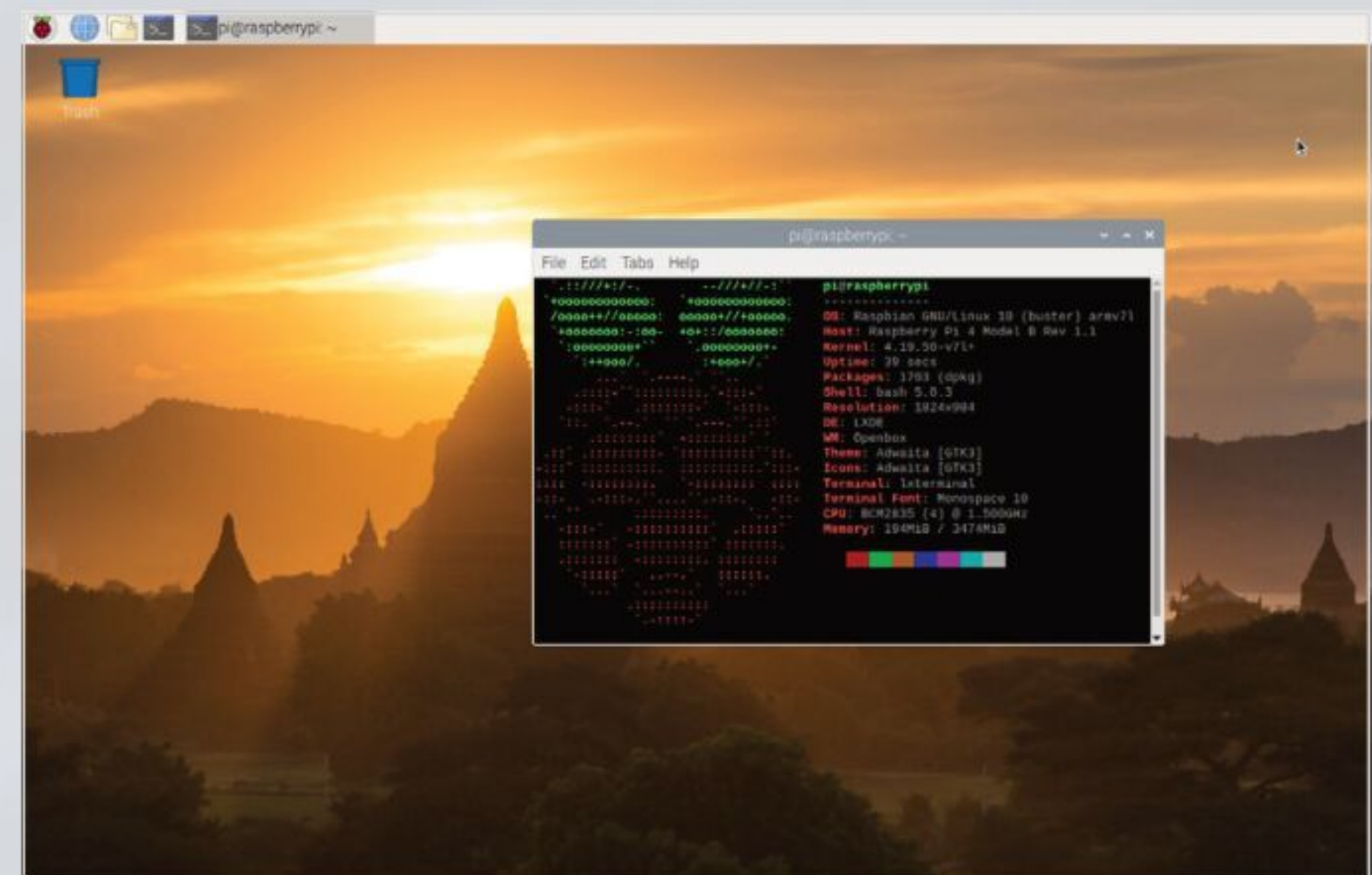
You'll also find a pair of USB 3.0 ports, Bluetooth 5.0, and a GPU that's capable of handling 4K resolutions and OpenGL ES 3.0 graphics. In short, the Pi 4 is the most powerful of the current Raspberry Pi models. However, the different memory versions have an increased cost. The 2GB version costs £34, 4GB is £54, and the 8GB Pi 4 will set you back £74. Remember to also factor in one or two micro-HDMI cables with your order.

RASPBIAN BUSTER

In addition to releasing the Pi 4, the Raspberry Pi team also compiled a new version of the Raspbian operating system, codenamed Buster.

In conjunction with the new hardware the Pi 4 boasts, Buster does offer a few updates. Although on the whole it's very similar in appearance and operation to the previous version of Raspbian. The updates are mainly in-line with the 4K's display and playback, giving the Pi 4 a new set of graphical drivers and performance enhancements.

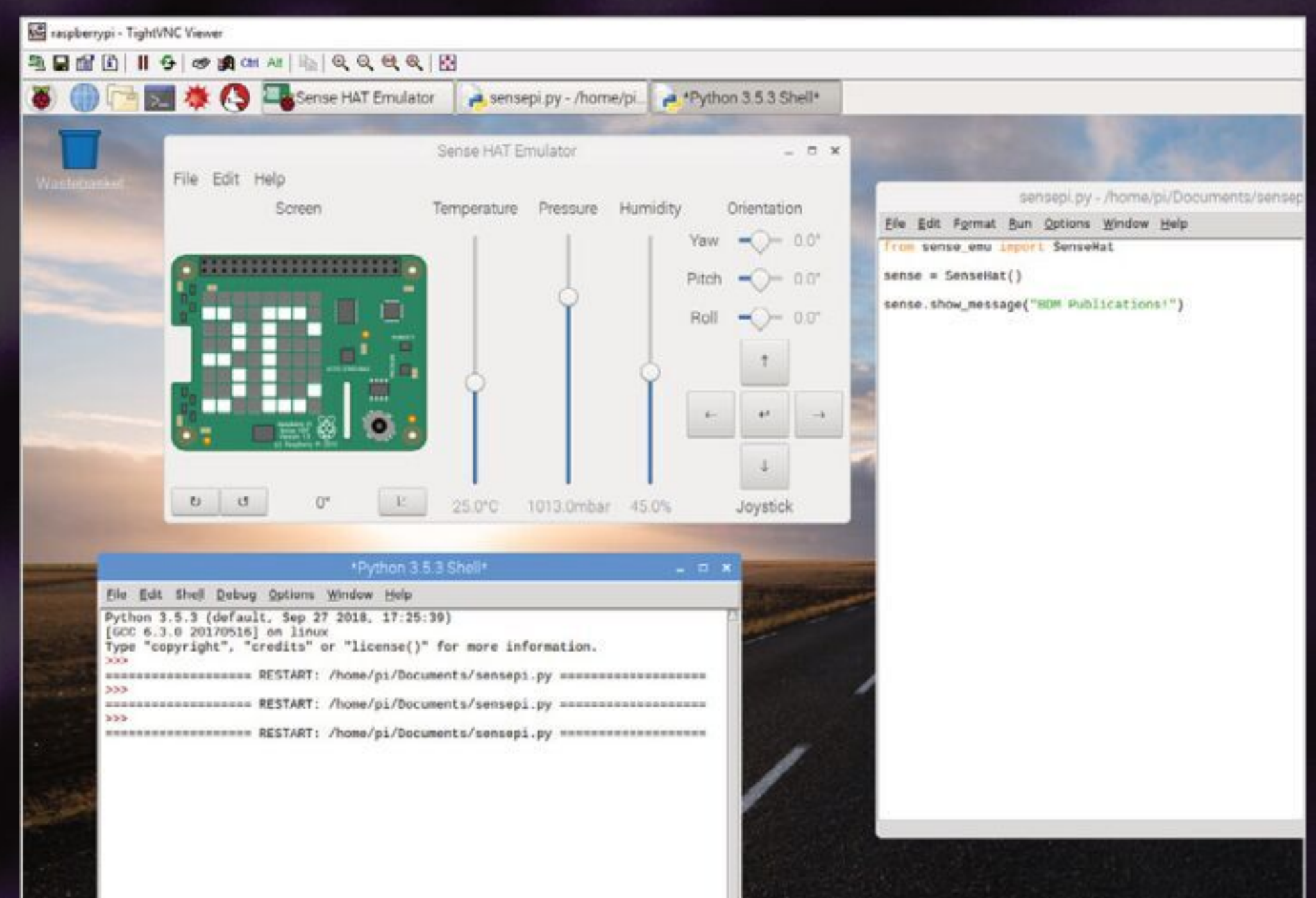
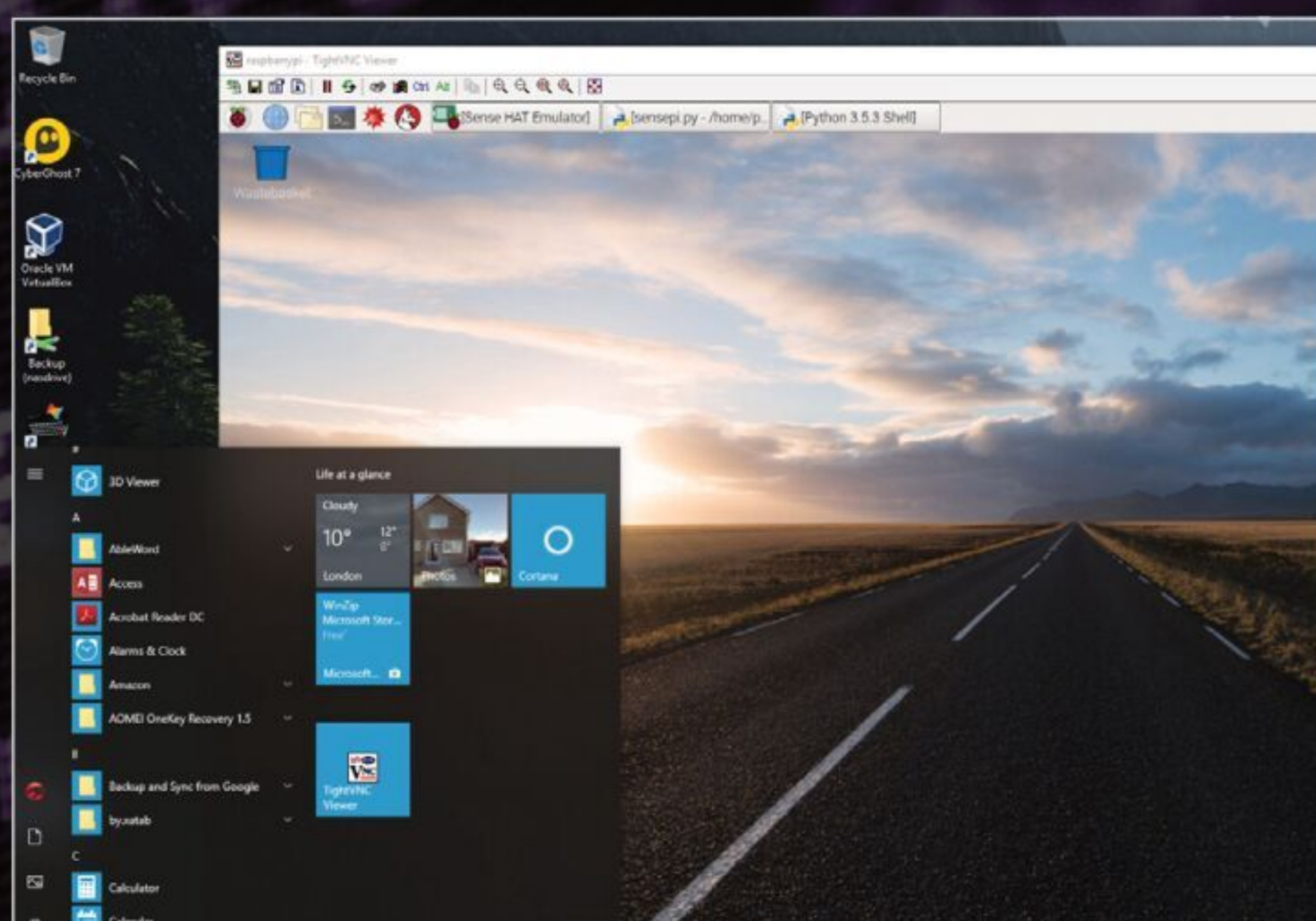
In short, what you see in this book will work with the Raspberry Pi 4 and Raspbian Buster!

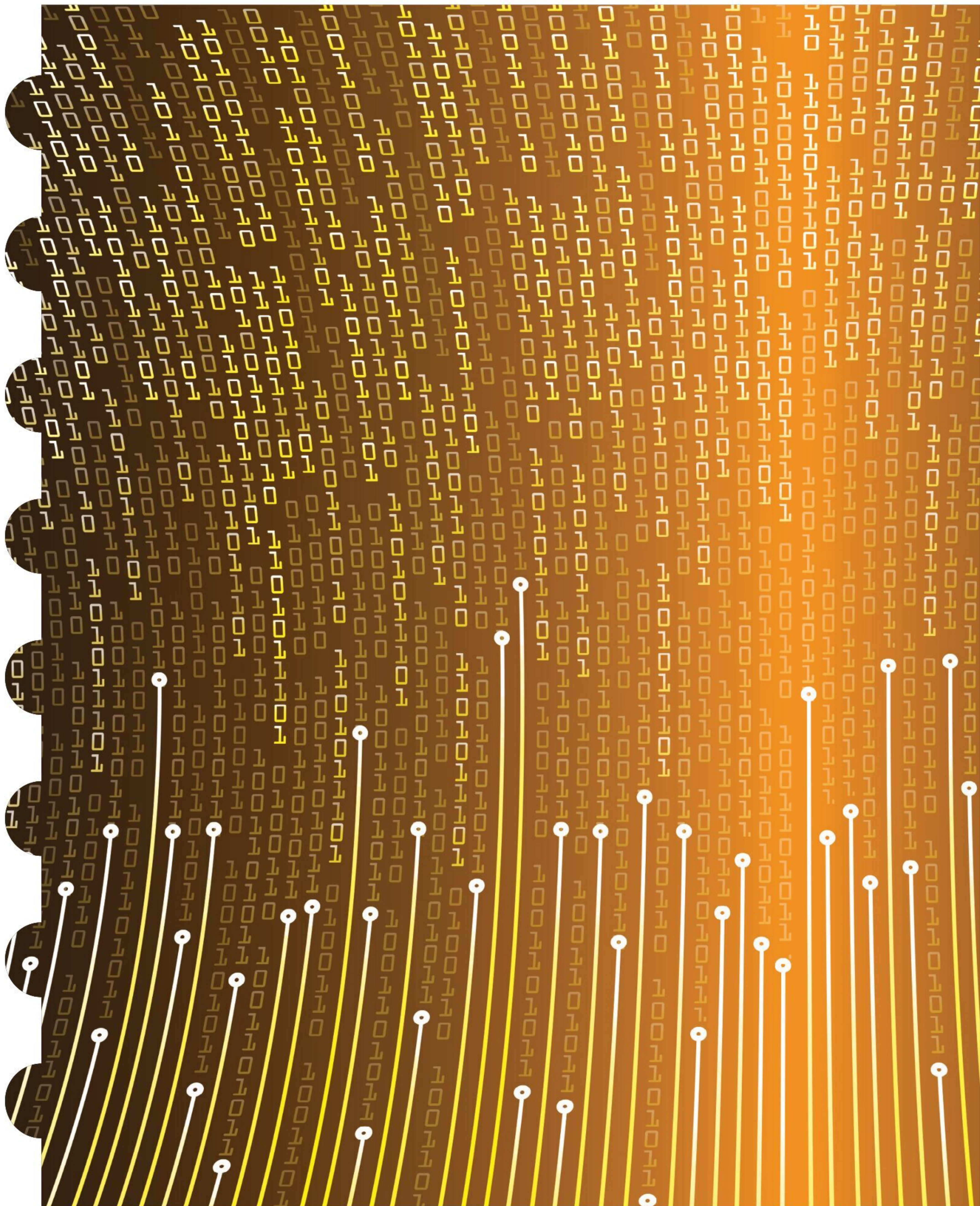


Once set up, you can remotely connect to the Pi's desktop from any device/PC.



You can even test connected hardware with Python remotely, via Windows.





Coding with Python

It's a long way from creating your first Python code to controlling a robot on Mars, but it's not impossible. Every advanced and professional Python coder started with the base fundamentals, tips and fixes mentioned in this section.

With these foundation Python concepts, you will be able to advance your skills and branch out into whatever Python project you have in mind. From printing Hello World on the screen, looking into conditions and loops, asking a user for input and reacting to it, to using variables to control your code, this is where your Python skills are forged.



Starting Python for the First Time

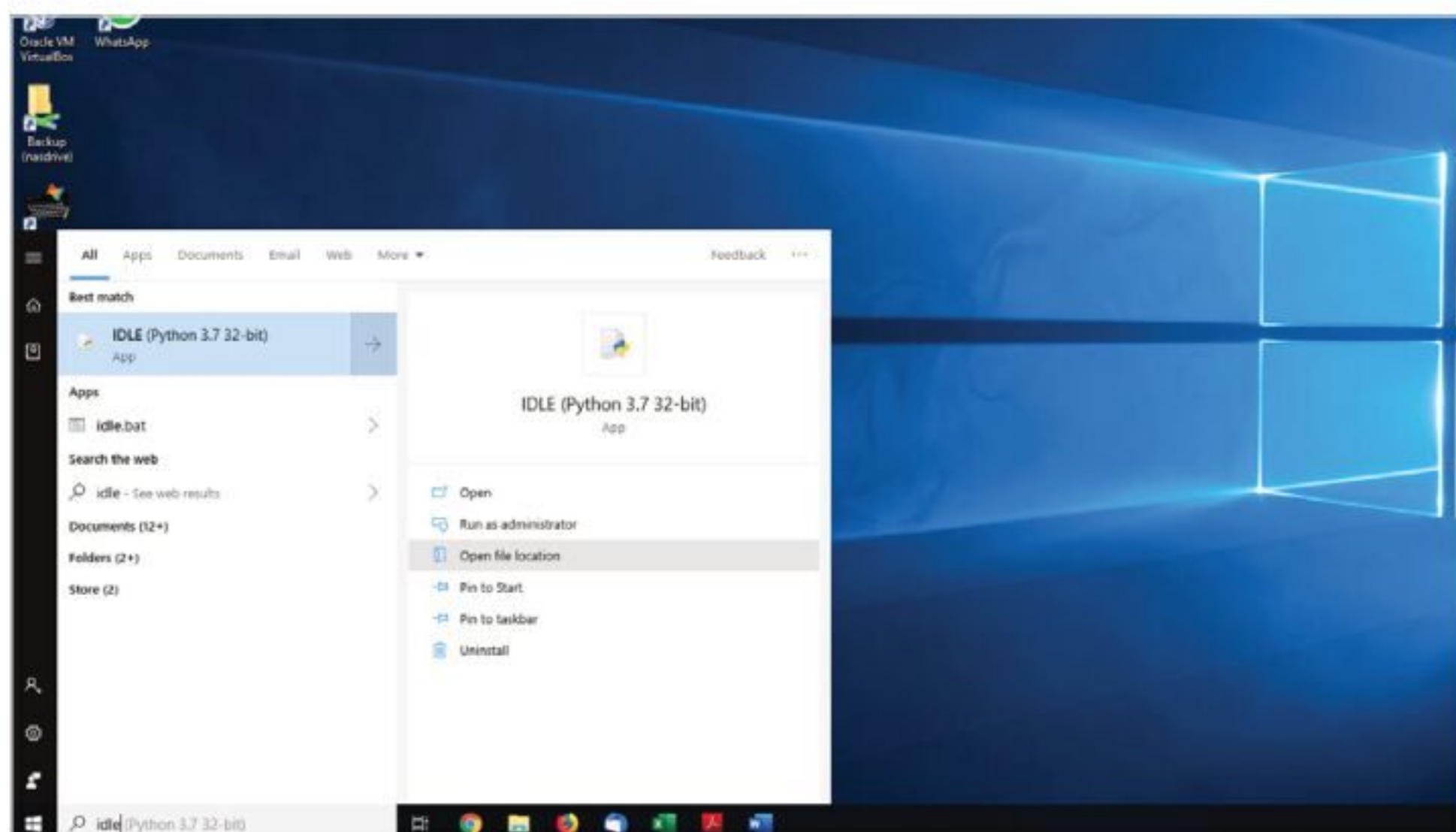
We're using Python 3 under Windows 10 for these following examples. Don't worry if your version of Python is 3.4.2, or something lesser than the current version, as long as you're using Python 3, the code will work.

STARTING PYTHON

As when learning anything new, you need to start slow. You can pick up the pace as your experience grows, but for now, let's just get something appearing on the screen. Don't worry, you'll soon be coding like a pro!

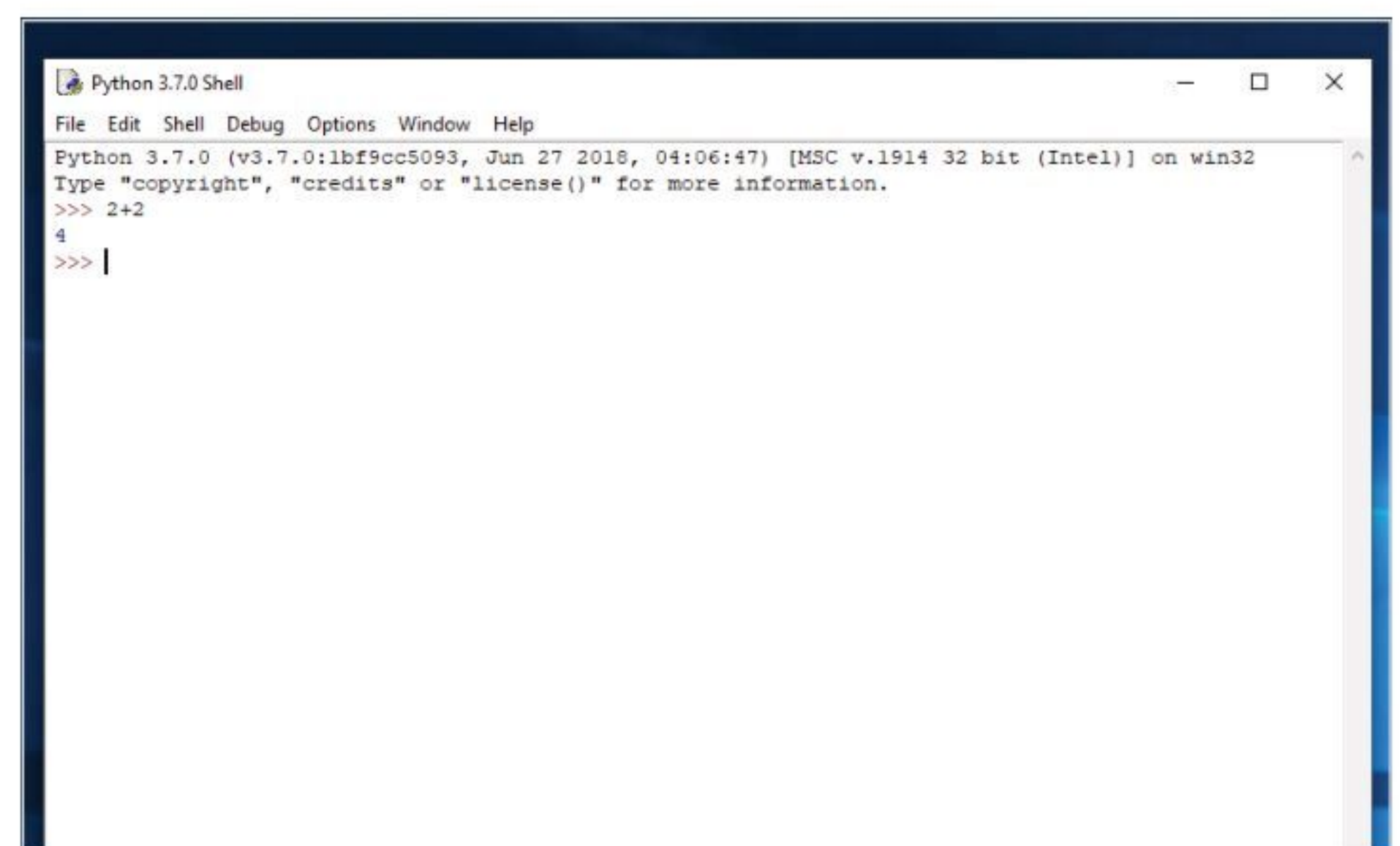
STEP 1

Click on the Windows Start button, and start typing 'idle'. The result will be the currently installed version of the Python IDLE on your system. You can Pin it to the Start for convenience, otherwise simply click the icon to launch the Python Shell.



STEP 3

For example, in the Shell enter: `2+2`
After pressing **Enter**, the next line will display the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.



STEP 2

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



STEP 4

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



STEP 5

While that's very interesting, it's not particularly exciting. Instead, try this:

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

Just enter it into the IDLE as you've done in the previous steps.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+645545522
645568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>> |
```

STEP 6

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

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

STEP 7

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

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands

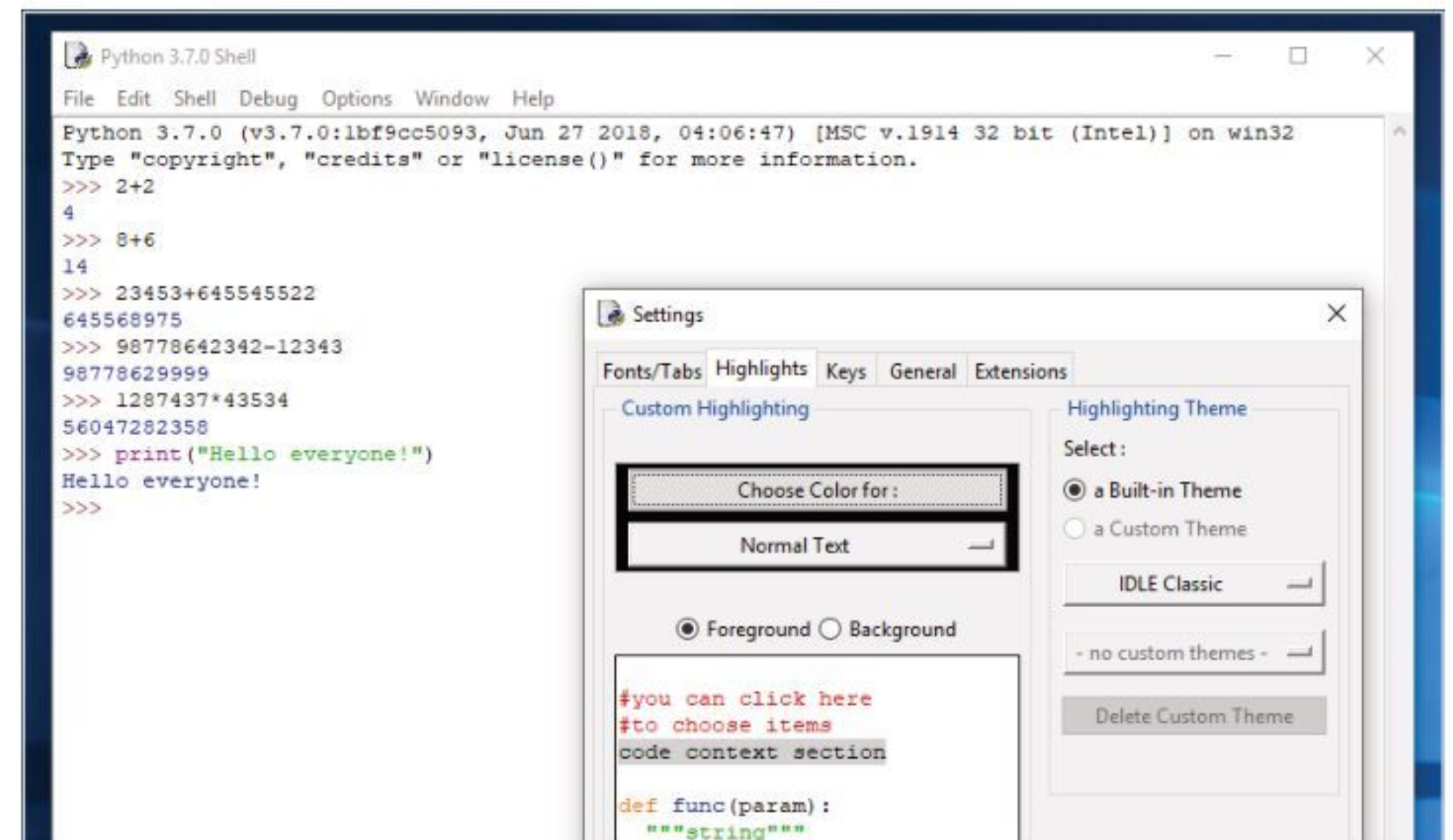
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

IDLE Colour Coding

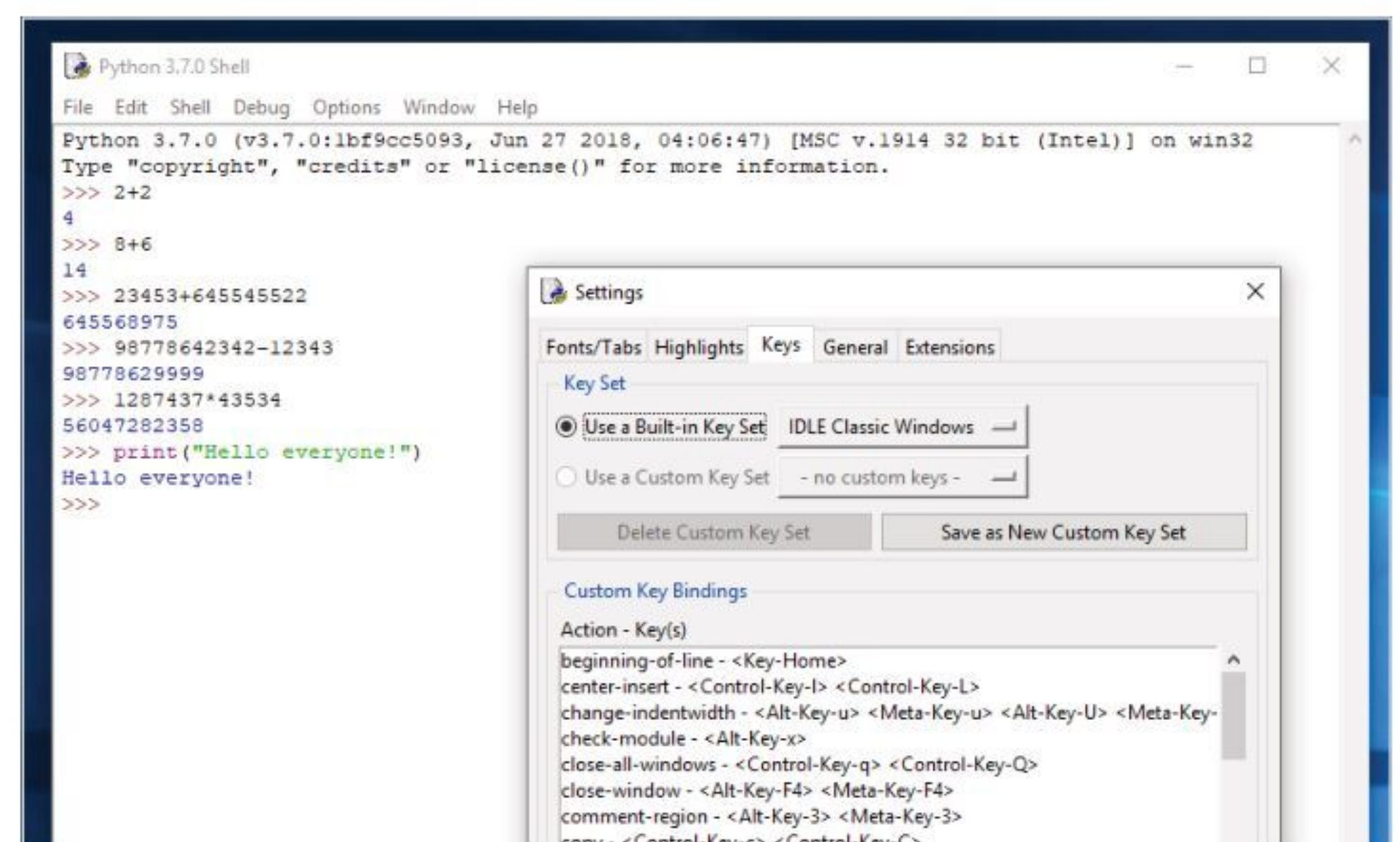
Colour	Use for	Examples
Black	Data & variables	23.6 area
Green	Strings	"Hello World"
Purple	Functions	len() print()
Orange	Commands	if for else
Blue	User functions	get_area()
Dark red	Comments	#Remember VAT
Light red	Error messages	SyntaxError:

STEP 8

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

**STEP 9**

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

**STEP 10**

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

25.5. IDLE

Source code: [Lib/Idlelib/](#)

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tcl/tk` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit window currently have the same top menu as Editor windows but a different default file and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)**New File**

Create a new file editing window

Open...

Open an existing file with an Open dialog

Recent Files

Open a list of recent files. Click one to open it

Open Module...

Open an existing module (searches sys path)

Class Browser

Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first



Your First Code

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

PLAYING WITH PYTHON

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

STEP 1 If you've closed Python 3 IDLE, re-open it as you did in the previous page. In the Shell, enter the familiar following:

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>>
```

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>>
```

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
```

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>>
```


STEP 5

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>>
```

STEP 6

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

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

This code takes the value of both a and b, adds them together, and outputs the result.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>>
```

STEP 7

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> name="David"
>>> print(name)
David
>>>
```

STEP 8

Now let's add a surname:

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

We now have two variables containing both a first name and a surname, and we can print them independently.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>>
```

STEP 9

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

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>>
```

STEP 10

In Python 3 we can separate the two variables with a space by using a comma:

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

Alternatively, you can add the space yourself:

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> print(name, surname)
David Hayward
>>>
```




Saving and Executing Your Code

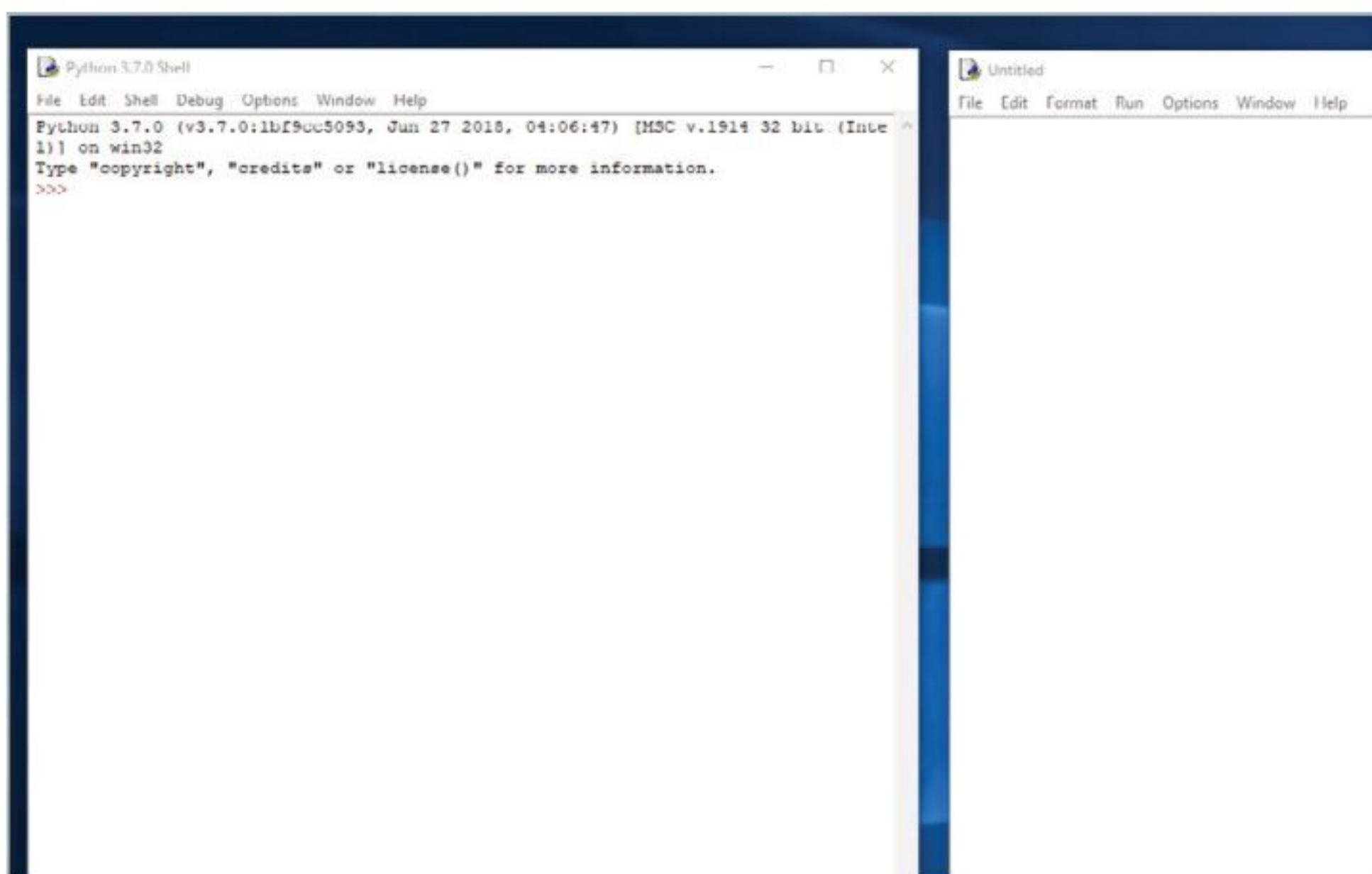
While working in the IDLE Shell is perfectly fine for snippets of code, it's not designed for entering longer program listings. In this section, we'll introduce you to the IDLE Editor, where most of our code will be entered from now on.

EDITING CODE

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

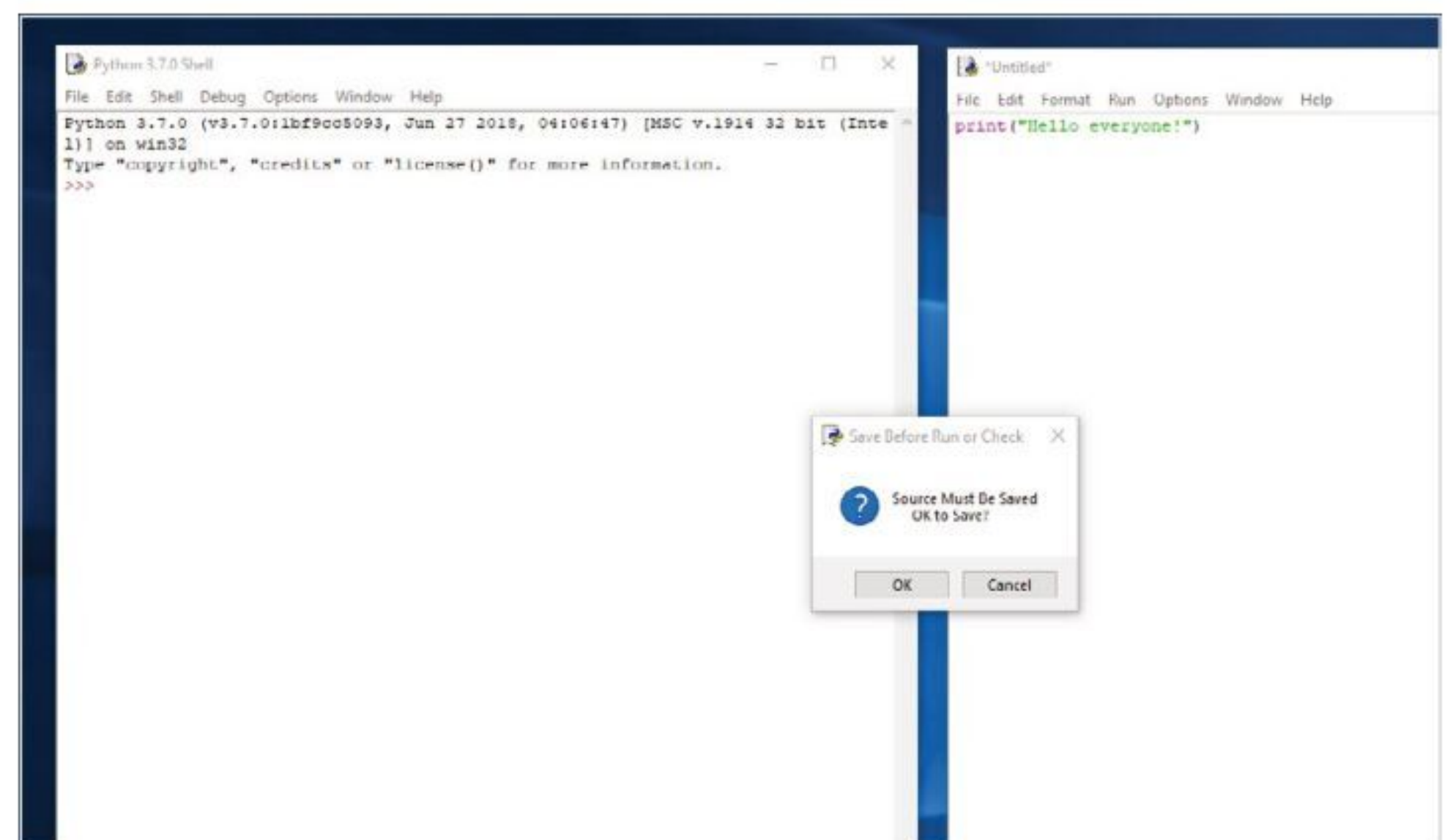
STEP 1

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



STEP 3

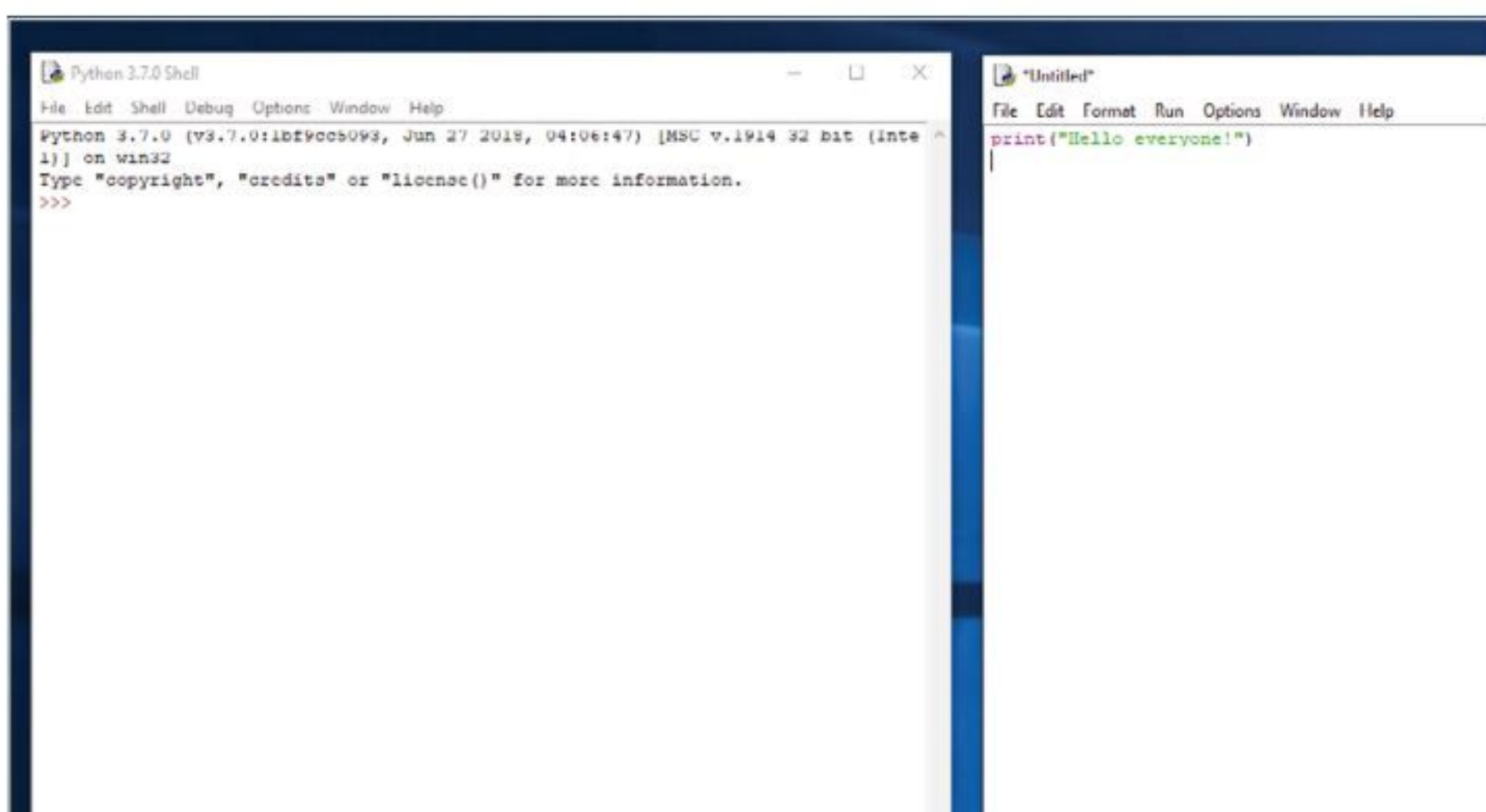
As you can see the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. To execute the code, however, you need to first save it. Press **F5** and you'll have a Save...Check box open.



STEP 2

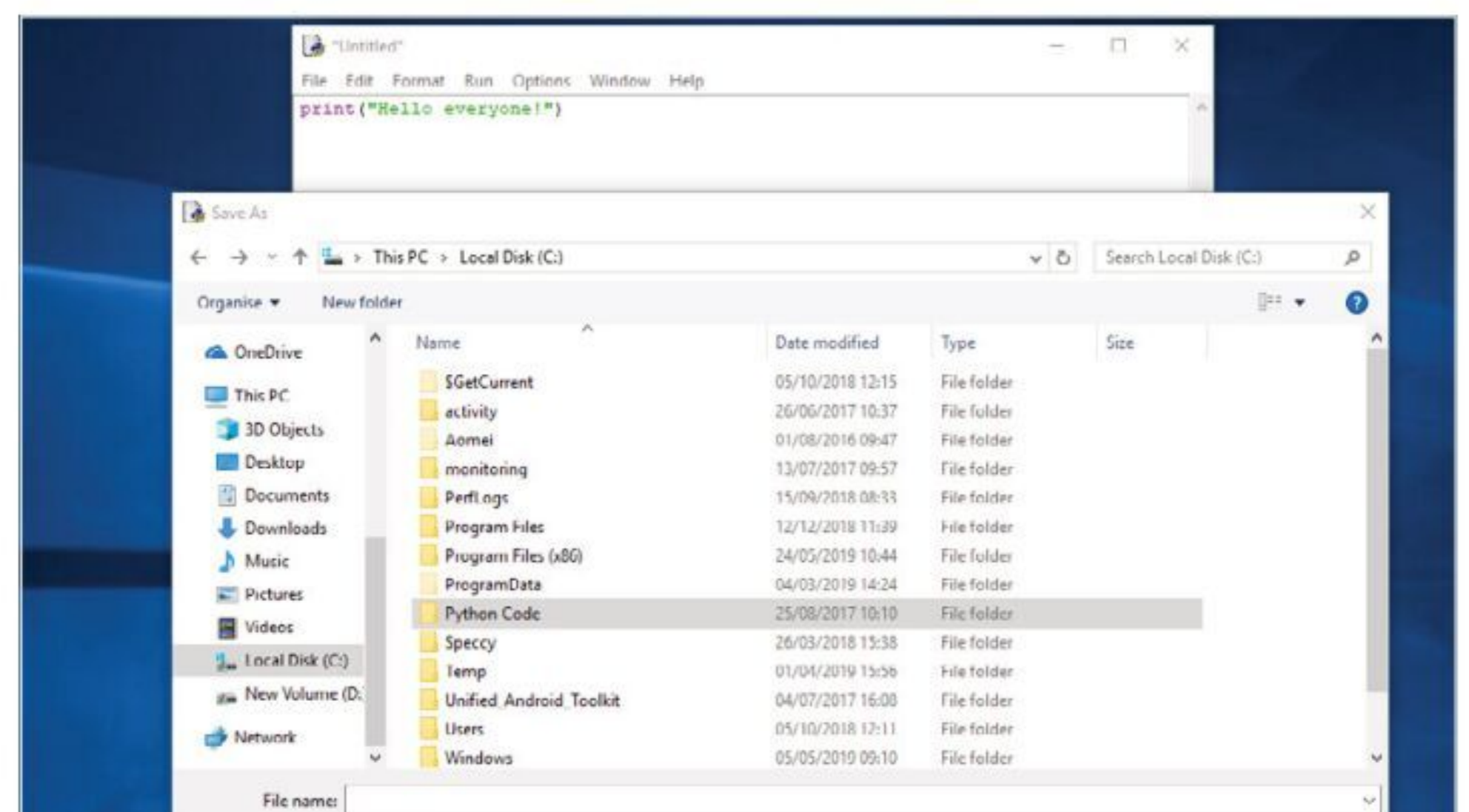
The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

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



STEP 4

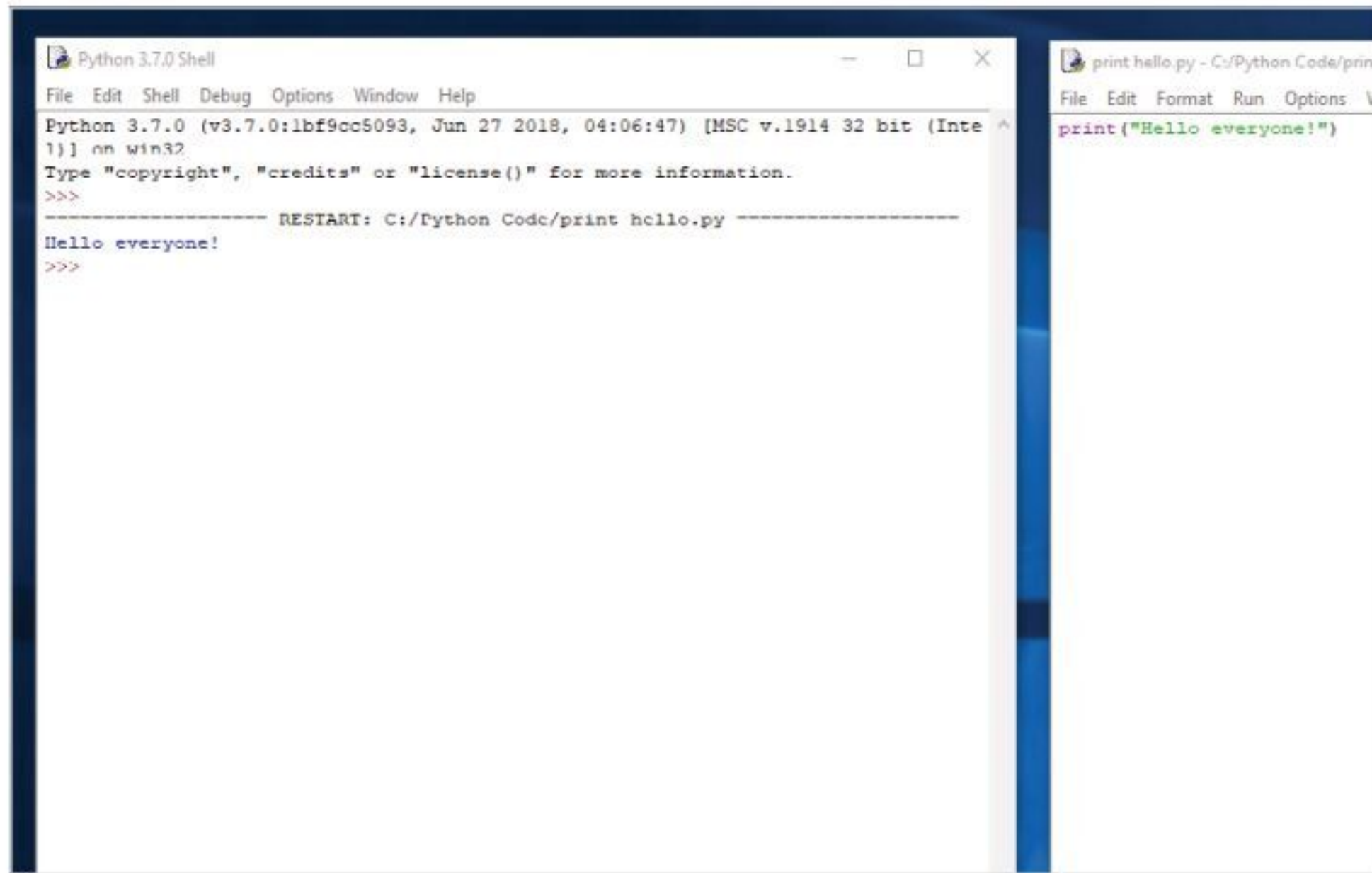
Click on the **OK** button in the Save box, and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python, or you can just dump it wherever you like. Remember to keep a tidy file system, though, it'll help you out in the future.





STEP 5

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



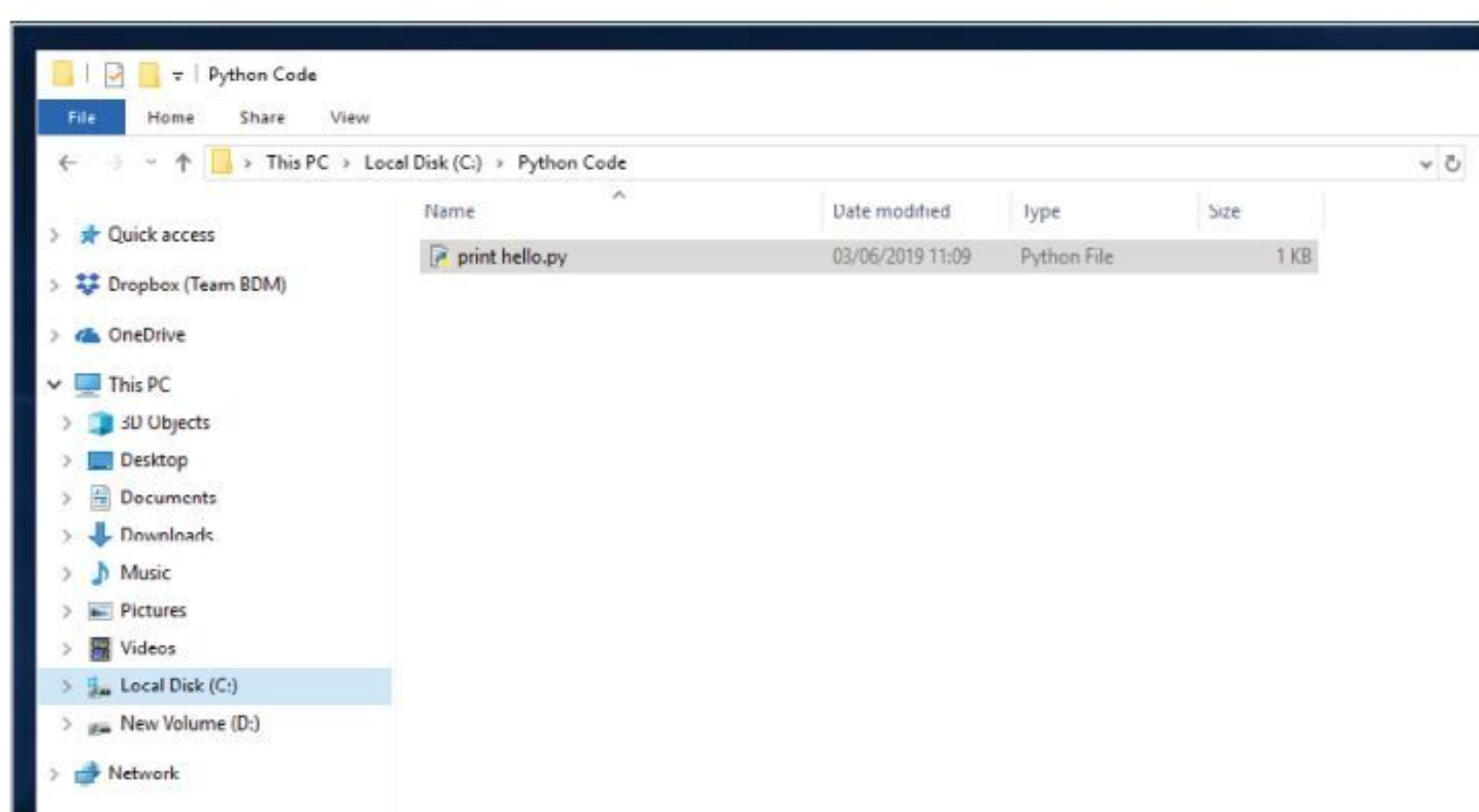
STEP 6

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



STEP 7

If you open the file location of the saved Python code, you'll notice that it ends in a .py extension. This is the default Python filename, any code you create will be whatever.py, and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.

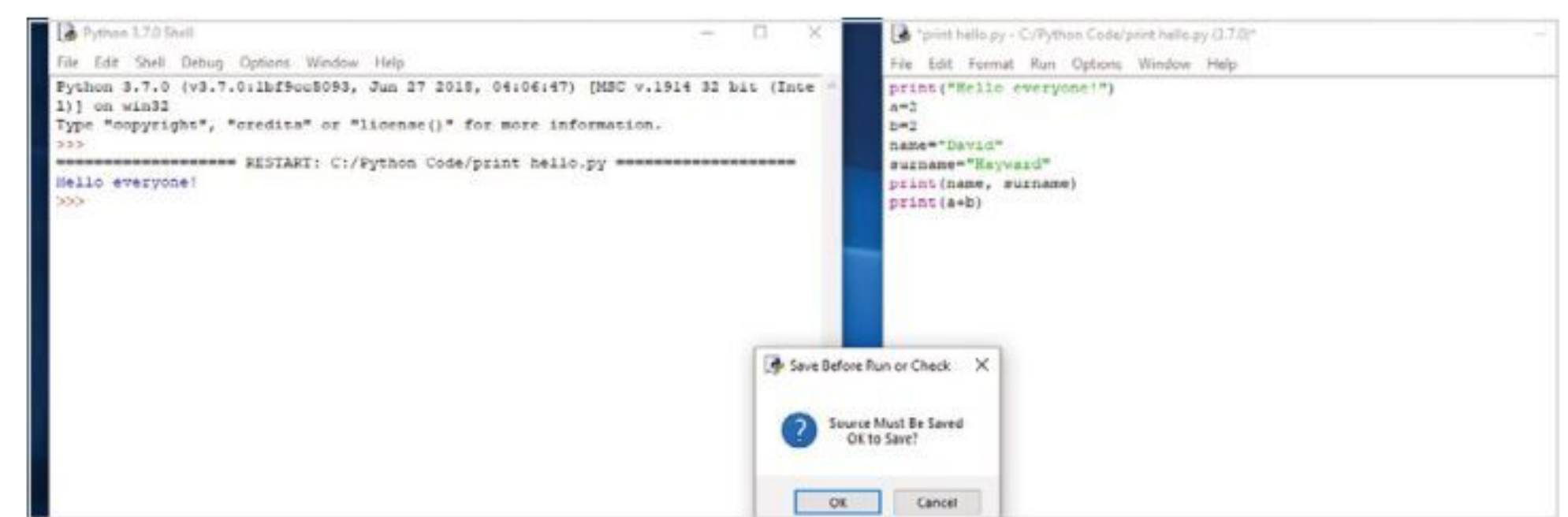


STEP 8

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

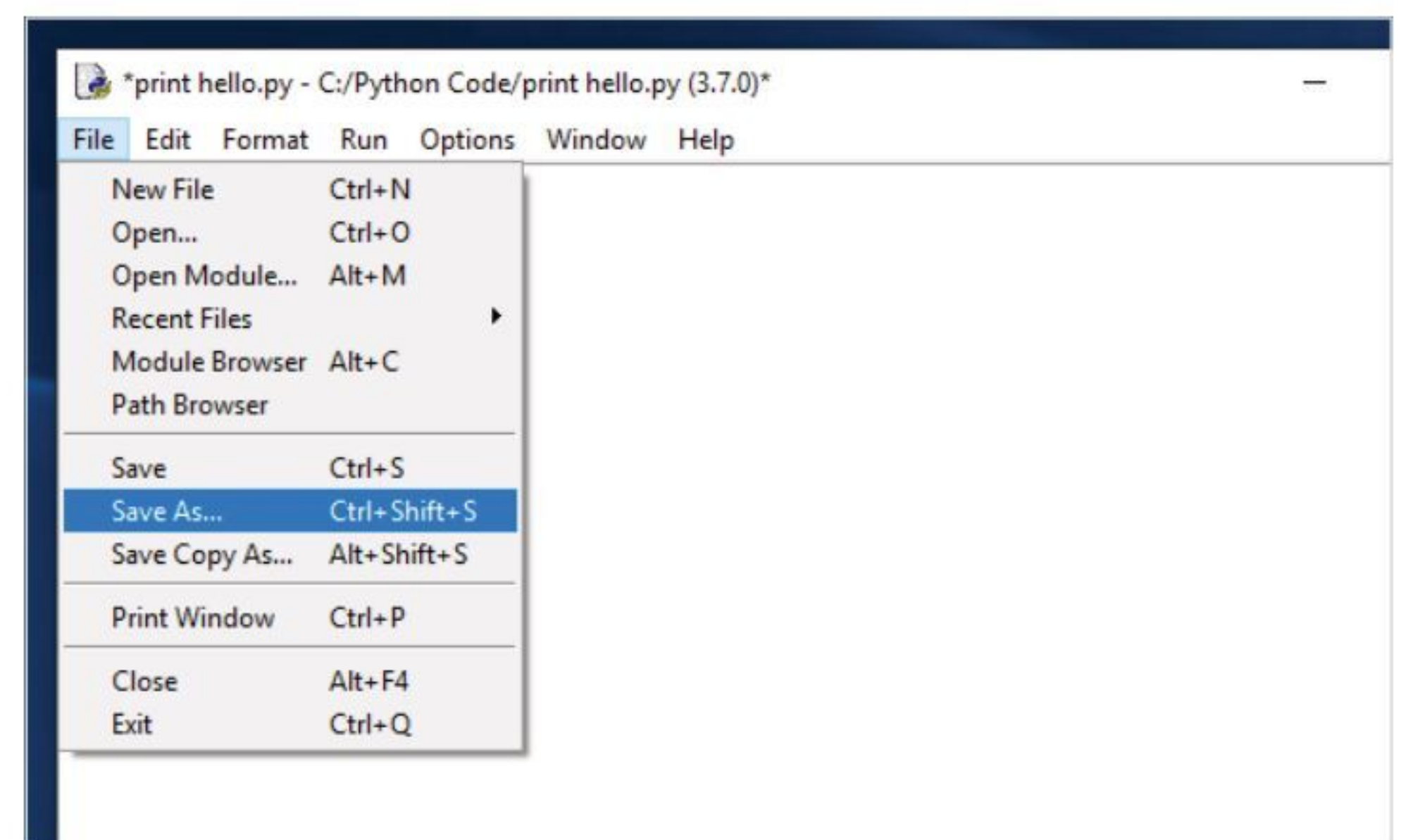
```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print(a+b)
```

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



STEP 9

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

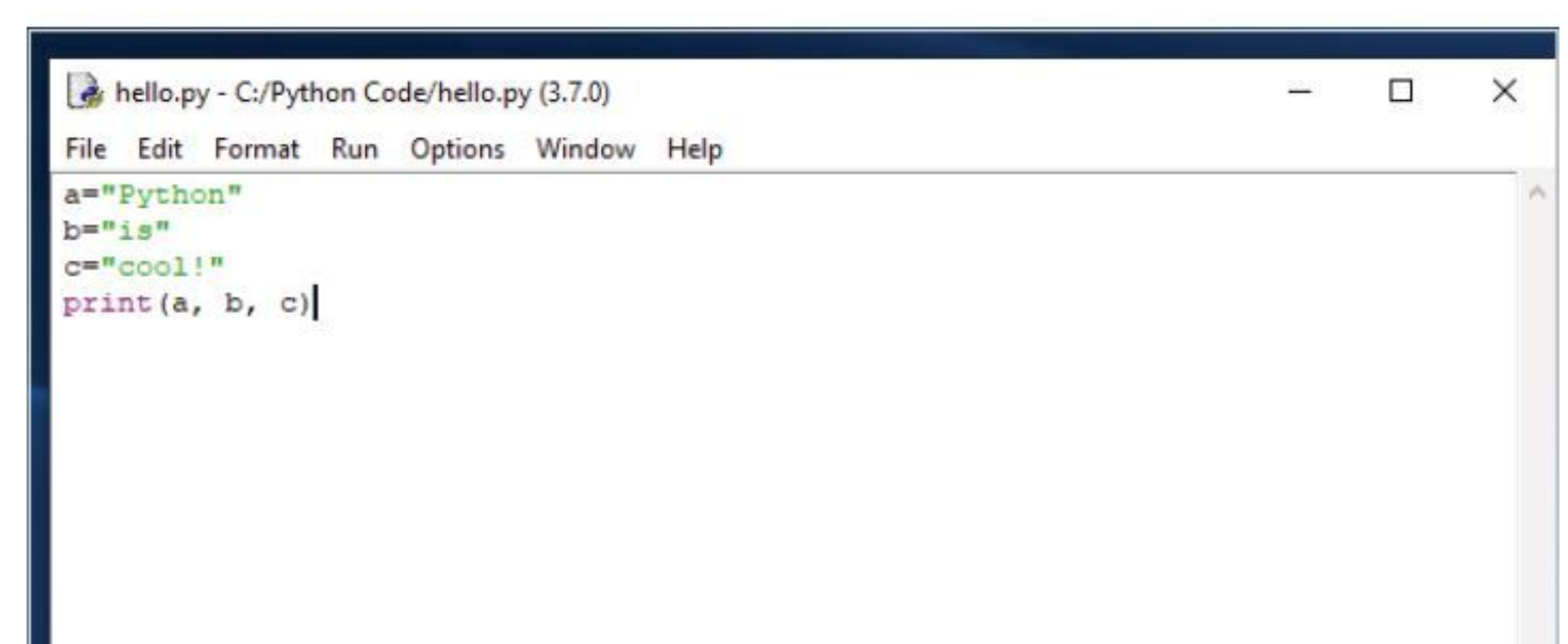


STEP 10

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

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

We will use this code in the next tutorial.





Executing Code from the Command Line

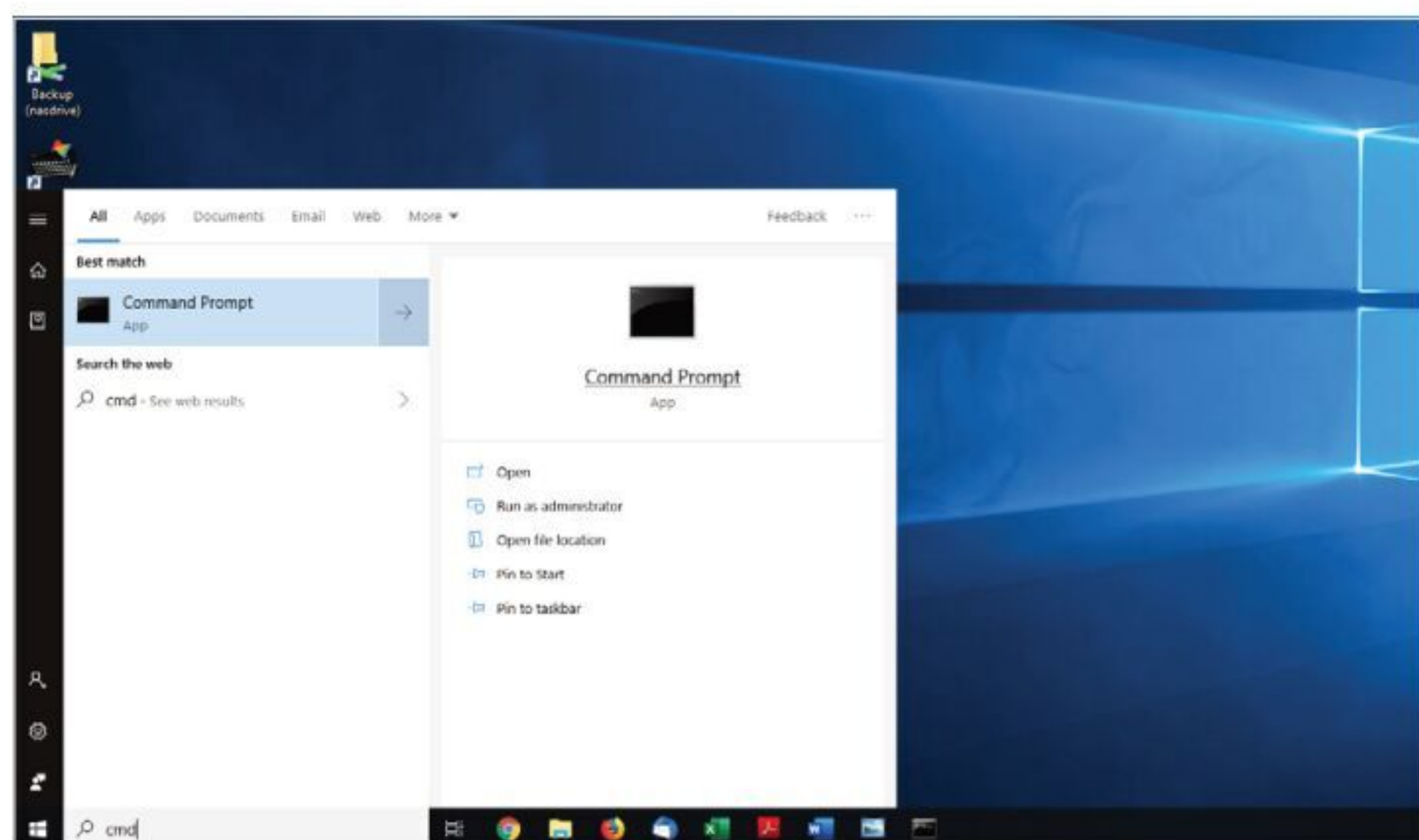
While we're going to be working from the GUI IDLE, it's worth taking a moment to look at Python's command line handling. Sometimes, depending on the code you write, executing via the command line is a better solution over the IDLE.

COMMAND THE CODE

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

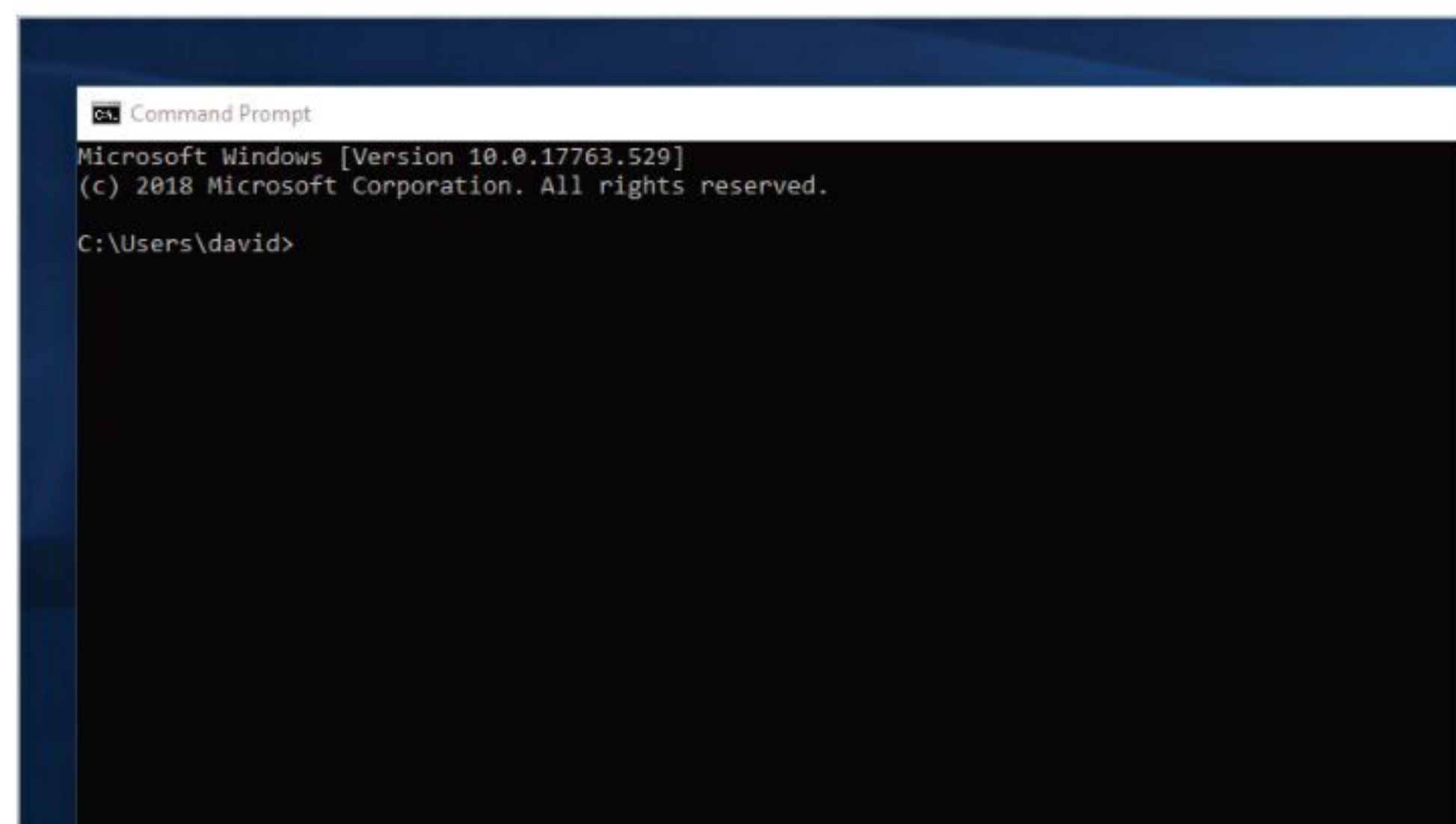
STEP 1

When you first installed Python, the installation routine automatically included all the necessary components to allow the execution of code outside of the GUI IDLE; in other words, the command line. To begin with, click on the Windows Start Button, and type: `cmd`.



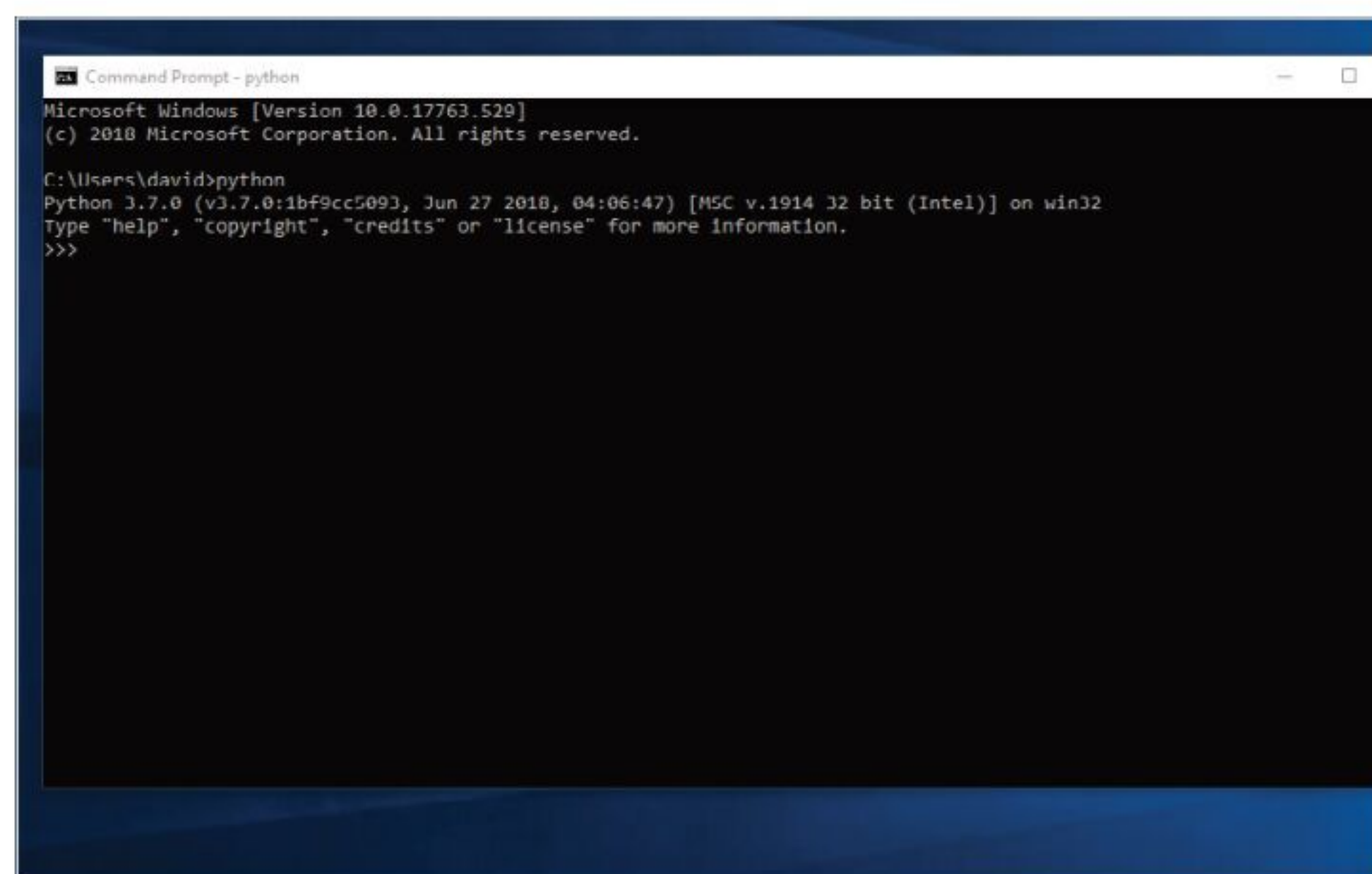
STEP 2

As you did when launching the Python IDLE, click on the returned result from the search, the Command Prompt App. This will launch a new window, with a black background and white text. This is the command line, also called a Terminal in macOS, Linux, and Raspberry Pi operating systems.



STEP 3

Now you're at the command line, we can start Python using the command `python` and pressing the Enter key. This will put you into the command line version of the Shell, with the familiar, three right-facing arrows as the cursor (`>>>`).

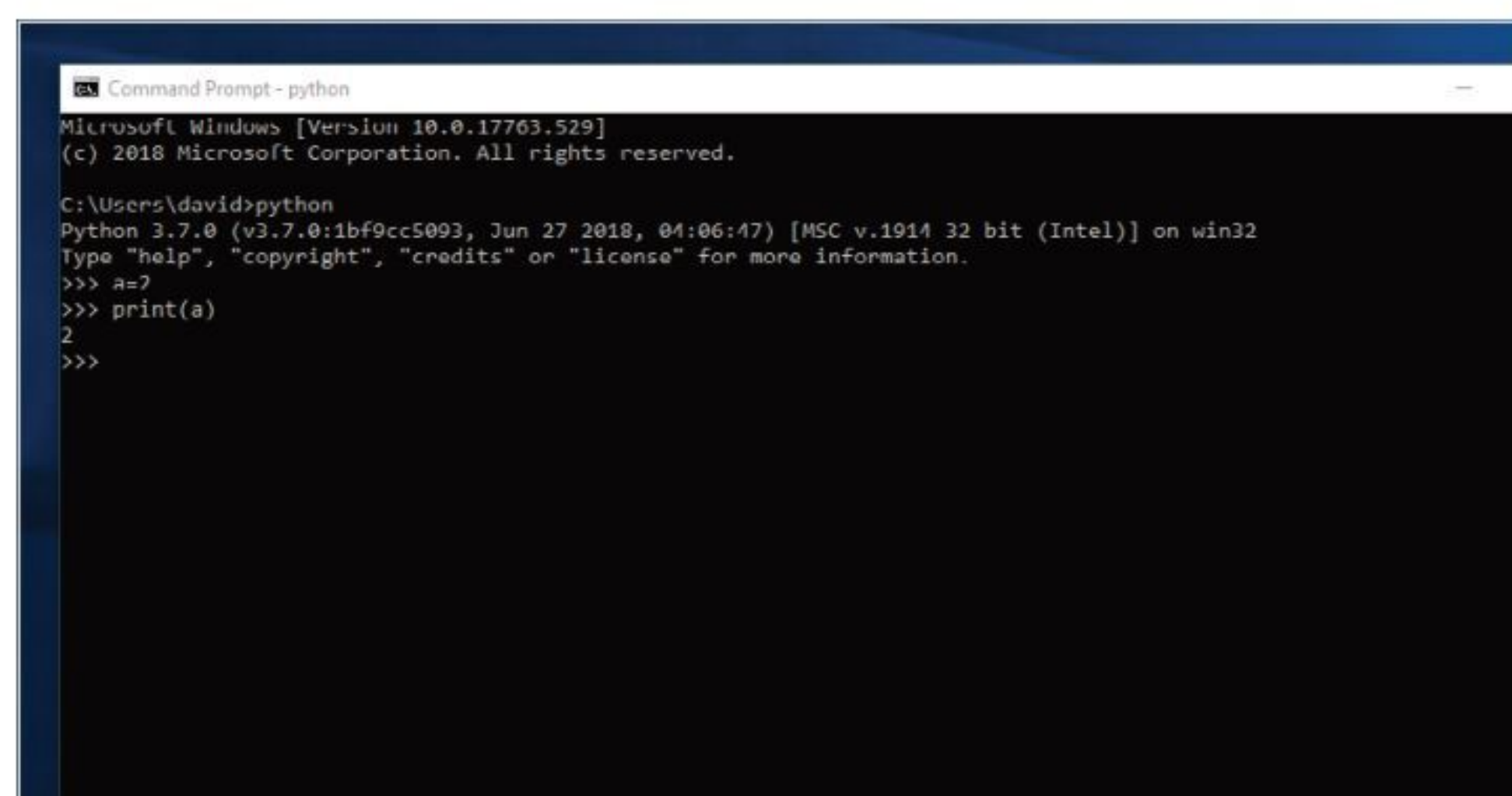


STEP 4

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

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

As you can see, it works exactly the same.



STEP 5

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

```

C:\Users\david>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=2
>>> print(a)
2
>>> exit()

C:\Users\david>cd \
C:\>cd "Python Code"

C:\Python Code>dir/w
Volume in drive C has no label.
Volume Serial Number is 8E47-ABFF

Directory of C:\Python Code

[.]                [..]                hello.py            print hello.py
                2 File(s)                73 bytes
                2 Dir(s)  76,514,889,728 bytes free

C:\Python Code>

```

STEP 6

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

```
python hello.py
```

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

```

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

```

```

C:\Python Code>python hello.py
Python is cool!

C:\Python Code>

```

DIFFERENT VERSIONS OF PYTHON

If you've previously used Python 3 on a Mac or Linux, and subsequently the Raspberry Pi, you may be a little confused as to why the Windows version of Python uses the command line: **python**, instead of **python3**.

The reason behind this is that UNIX-like systems, such as macOS and Linux, already have Python libraries pre-installed. These older libraries are present because some of the macOS and Linux system utilities rely on Python 2, and therefore installing a newer version of Python, and thus altering the executable name, could have dire consequences to the system.

As a result, developers decided that the best approach for macOS and Linux systems would be to leave the command line '**python**' as exclusive Python 2 use, and newer versions of user-installed Python would be '**python3**'.

This isn't an issue with Windows, as it doesn't use any Python libraries other than the ones installed by the user themselves when actually installing Python. When a Windows user installs Python, the installation wizard will auto-include the command line instance to the core Windows PATH variable, which you can view by entering: `path` into the command line. This points to the python.exe file required to execute Python code from the command line.

We don't recommend you install both Python 2 and Python 3 within Windows 10; naturally, you can if you want, but realistically, although Python 2 still has a foothold in the coding world, Python 3 is the newest version. However, if you do, then you will need to rename one of the Python versions names; as they will be installed in different folders and both use python.exe as the command line executable. It's a little long-winded, so unless there's a dire need to have both versions of Python installed, it's best to stick to Python 3.

```

C:\Users\david>path
PATH=C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\Shared;C:\Program Files\PuTTY\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Users\david\AppData\Local\Programs\Python\Python37-32\Scripts\;C:\Users\david\AppData\Local\Programs\Python\Python37-32\;C:\Users\david\AppData\Local\Microsoft\WindowsApps;C:\Users\david\AppData\Local\Programs\Python\Python36-32\;C:\Users\david\AppData\Local\Microsoft\WindowsApps

C:\Users\david>

```




Numbers and Expressions

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

IT'S ALL MATHS, MAN

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

STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, we're going to use the Shell. If you've opted to use a third-party text editor, note that you need to get to the IDLE Shell for this part of the tutorial.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 In the Shell enter the following:

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

As you can see, Python can handle some quite large numbers.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>>
```

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

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

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

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

and so on.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>>
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>>
```


STEP 5

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

```
10/3
```

will display 3.33333333, which is, of course, 3.3-recurring. If you now enter:

```
10%3
```

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 17:46:43) on win32
Type "copyright", "credits" or "license()" for more
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> |
```

STEP 6

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

```
2**3
```

```
10**10
```

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 17:46:43) on win32
Type "copyright", "credits" or "license()" for more
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> |
```

STEP 7

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

```
bin(3)
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 17:46:43) on win32
Type "copyright", "credits" or "license()" for more
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> |
```

STEP 8

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

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 17:46:43) on win32
Type "copyright", "credits" or "license()" for more
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3, 'b')
'11'
>>> |
```

STEP 9

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

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

```
Booleantest.py - C:/Python Code/Booleantest.py
File Edit Format Run Options Window
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```

STEP 10

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
>>>
===== RESTART: C:/Python Code/Booleantest.py =====
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>> |
```




Using Comments

When writing your code, the flow, what each variable does, how the overall program will operate and so on, is all inside your head. Another programmer could follow the code line by line, but when the code starts to hit thousands of lines, things get a little difficult to read.

#COMMENTS!

A method used by most programmers for keeping their code readable, is by commenting on certain sections. For example, if a variable is used, the programmer comments on what it's supposed to do. It's just good practise.

STEP 1

We'll start by creating a new instance of the IDLE Editor (**File > New File**), and then create a simple variable and print command:

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

Save the file, and execute the code.

```
Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

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

STEP 3

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/Comments.py =====
The value of A is, 10
>>>
===== RESTART: C:/Python Code/Comments.py =====
The value of A is, 10
>>>
```

STEP 2

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help

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

STEP 4

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help

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




STEP 5

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/Comments.py =====
The value of A is, 10
>>>
===== RESTART: C:/Python Code/Comments.py =====
The value of A is, 10
>>>
===== RESTART: C:/Python Code/Comments.py =====
The value of A is, 10
You've just lost a life!
You now have 9 lives left!
>>>
```

STEP 6

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 7

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help
# This is the best game ever, and has been developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite being very smelly, the code at least
# works really well.

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

STEP 8

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform player, and display current of A (lives)
```

STEP 9

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 10

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

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

```
*Comments.py - C:/Python Code/Comments.py (3.7.0)*
File Edit Format Run Options Window Help
'''
This is the best game ever, and has been developed by a crack squad of Python experts
who haven't slept or washed in weeks. Despite being very smelly, the code at least
works really well.
'''

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




Working with Variables

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

VARIOUS VARIABLES

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

STEP 1

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> |
```

STEP 3

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>> |
```

STEP 2

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> |
```

STEP 4

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

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

Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

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

Which, as we know, are integers.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>> character=name + ": " + title
>>> print(character)
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>> |
```


STEP 5

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

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

```
j on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>> character=name + ": " + title
>>> print(character)
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: can only concatenate str (not "int") to str
>>>
```

STEP 6

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

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

Alternatively, you can use:

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

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

```
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: can only concatenate str (not "int") to str
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Celts is 46 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Celts is 46 years old.
>>>
```

STEP 7

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

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> age=input("How old are you? ")
How old are you? 46
>>> type(age)
<class 'str'>
>>>
```

STEP 8

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

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> age=input("How old are you? ")
How old are you? 46
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: can only concatenate str (not "int") to str
>>> int(age) + 10
56
>>>
```

STEP 9

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

```
shirt=19.99
```

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>>
```

STEP 10

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>>
```




User Input

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

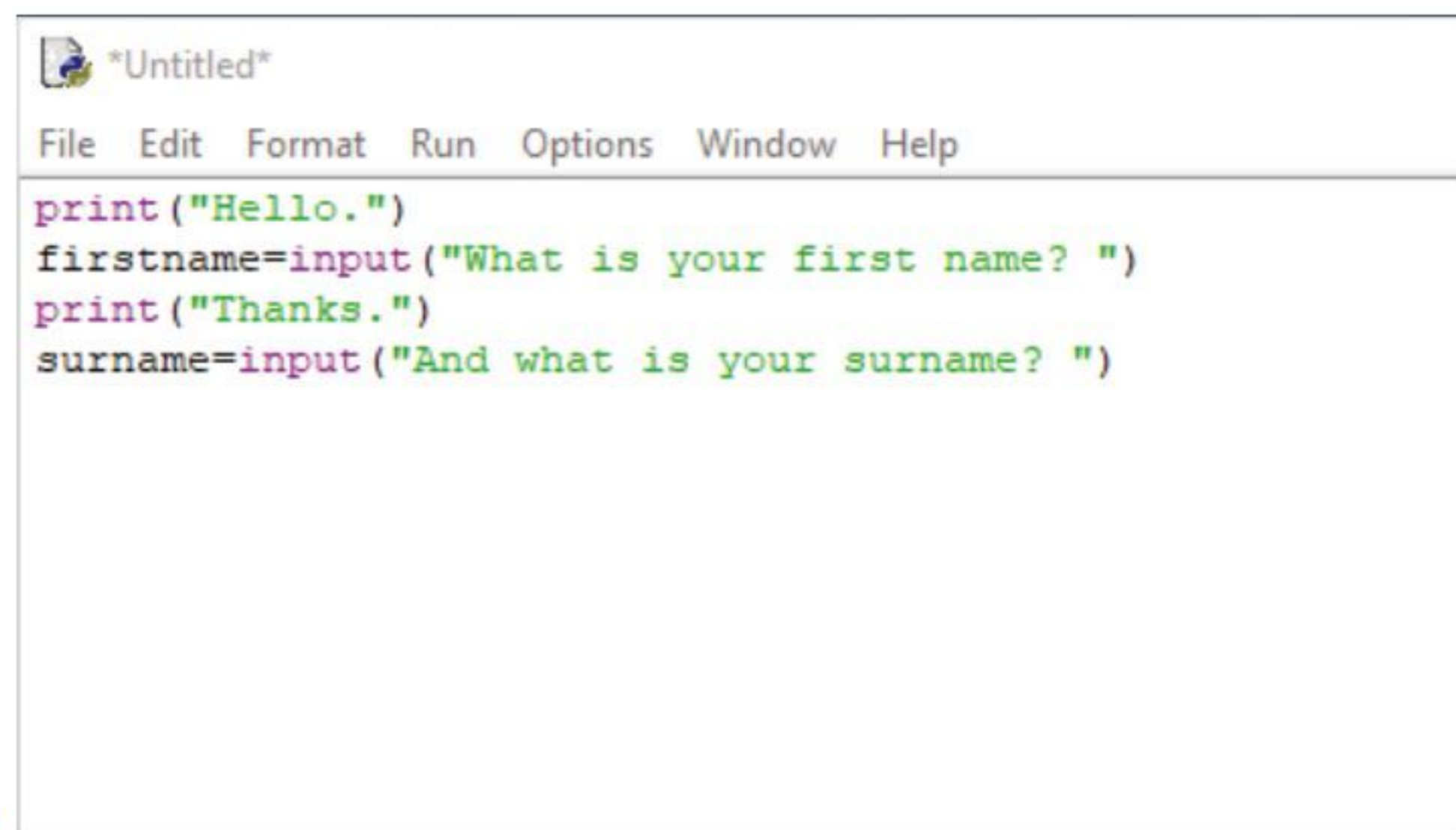
USER FRIENDLY

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

STEP 1

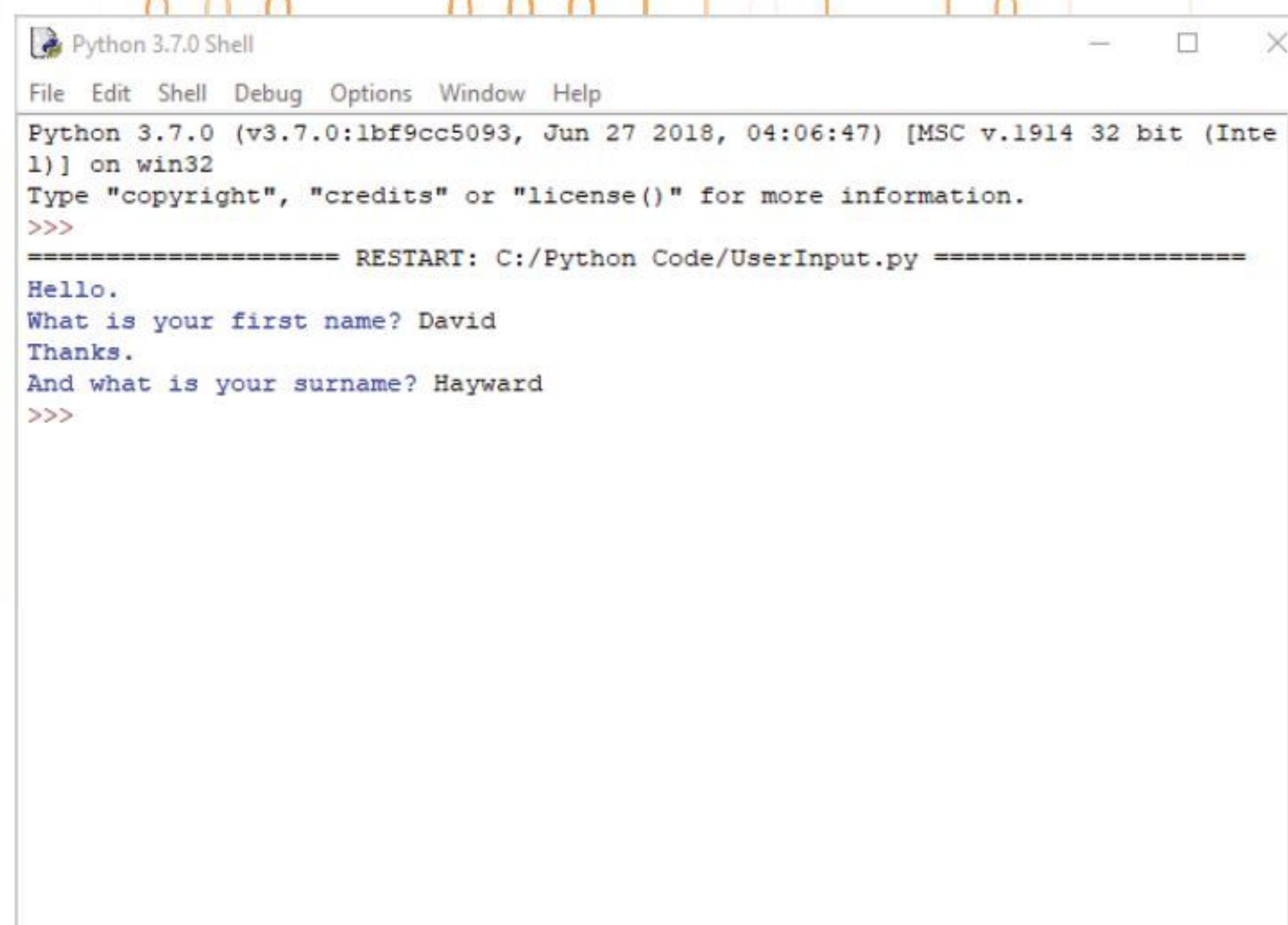
If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

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



STEP 2

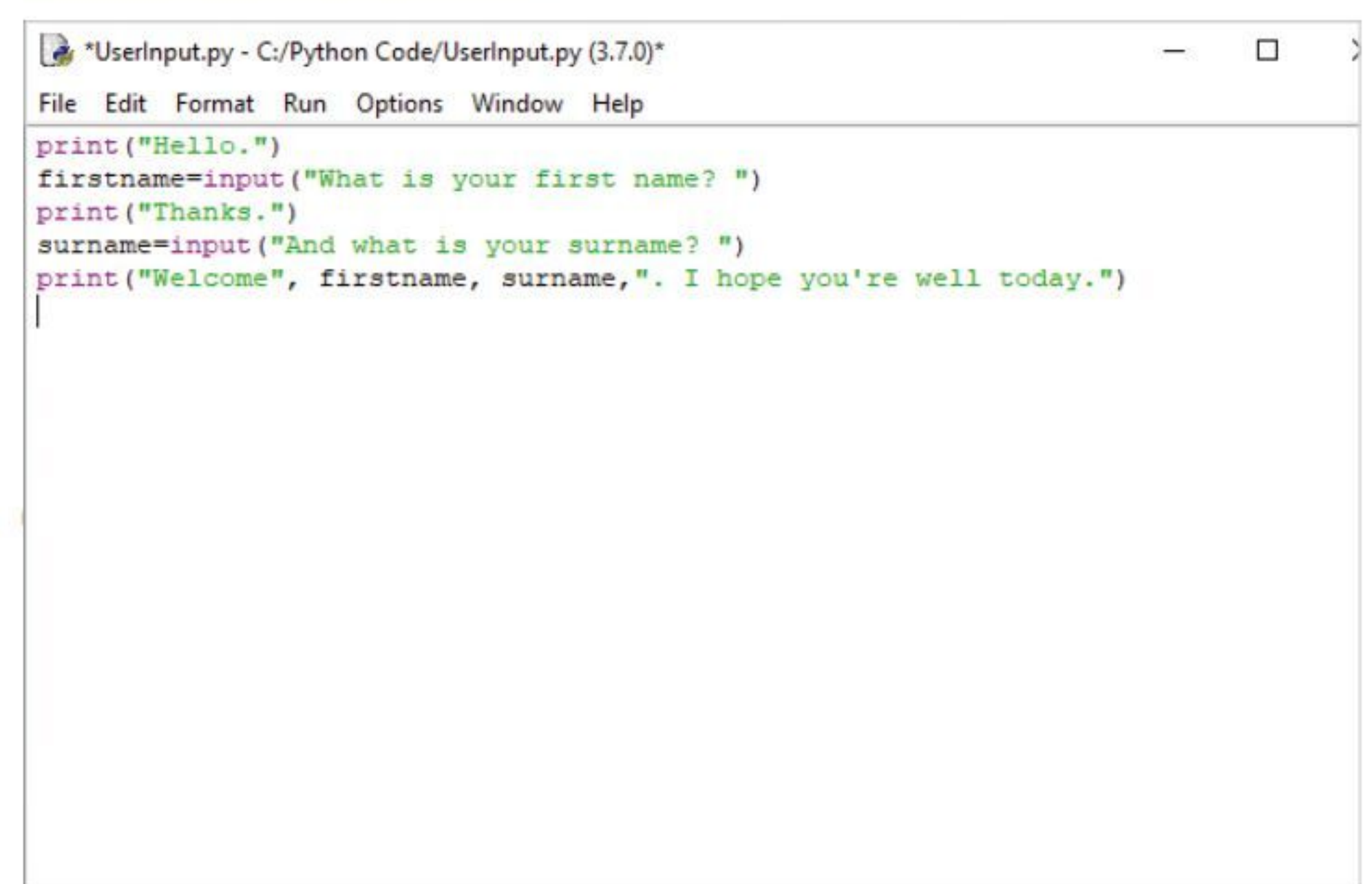
Save and execute the code, and, as you no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable **firstname**, followed by your surname; also stored in its own variable (**surname**).



STEP 3

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

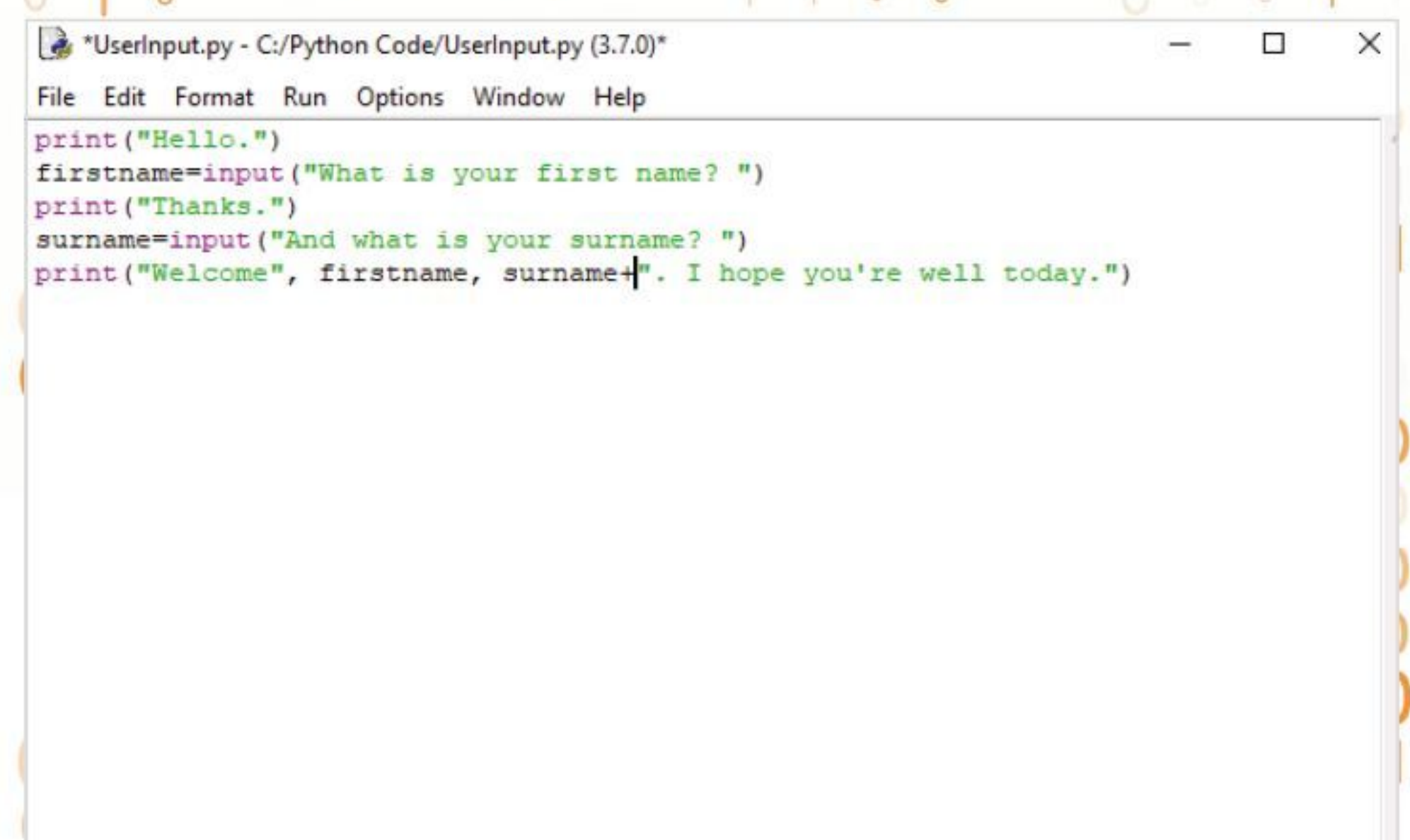
```
print("Welcome", firstname, surname, ". I hope you're well today.")
```



STEP 4

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

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



STEP 5

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

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

```
*UserInput.py - C:/Python Code/UserInput.py (3.7.0)*
File Edit Format Run Options Window Help
print("Hello. What is your name?")
name=input()
```

STEP 8

What you've created here is a condition, based on the user's input. In short, we're using the input from the user and measuring it against a condition. Therefore, if the user enters David as their name, the guard will allow them to pass unhindered. If, however, they enter a name other than David, the guard challenges them to a fight.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/UserInput.py =====
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>>
===== RESTART: C:/Python Code/UserInput.py =====
Halt! Who goes there?
Jim
I know you not. Prepare for battle!
>>>
```

STEP 6

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

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

```
*UserInput.py - C:/Python Code/UserInput.py (3.7.0)*
File Edit Format Run Options Window Help
print("Halt! Who goes there?")
name=input()
|
```

STEP 9

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

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

```
*UserInput.py - C:/Python Code/UserInput.py (3.7.0)*
File Edit Format Run Options Window Help
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```

STEP 7

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

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

```
*UserInput.py - C:/Python Code/UserInput.py (3.7.0)*
File Edit Format Run Options Window Help
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
|
```

STEP 10

And to finalise the rate and distance code, we can add:

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/UserInput.py =====
Input a rate and a distance
Rate: 12
Distance: 24
Time: 2.0
>>>
```




Creating Functions

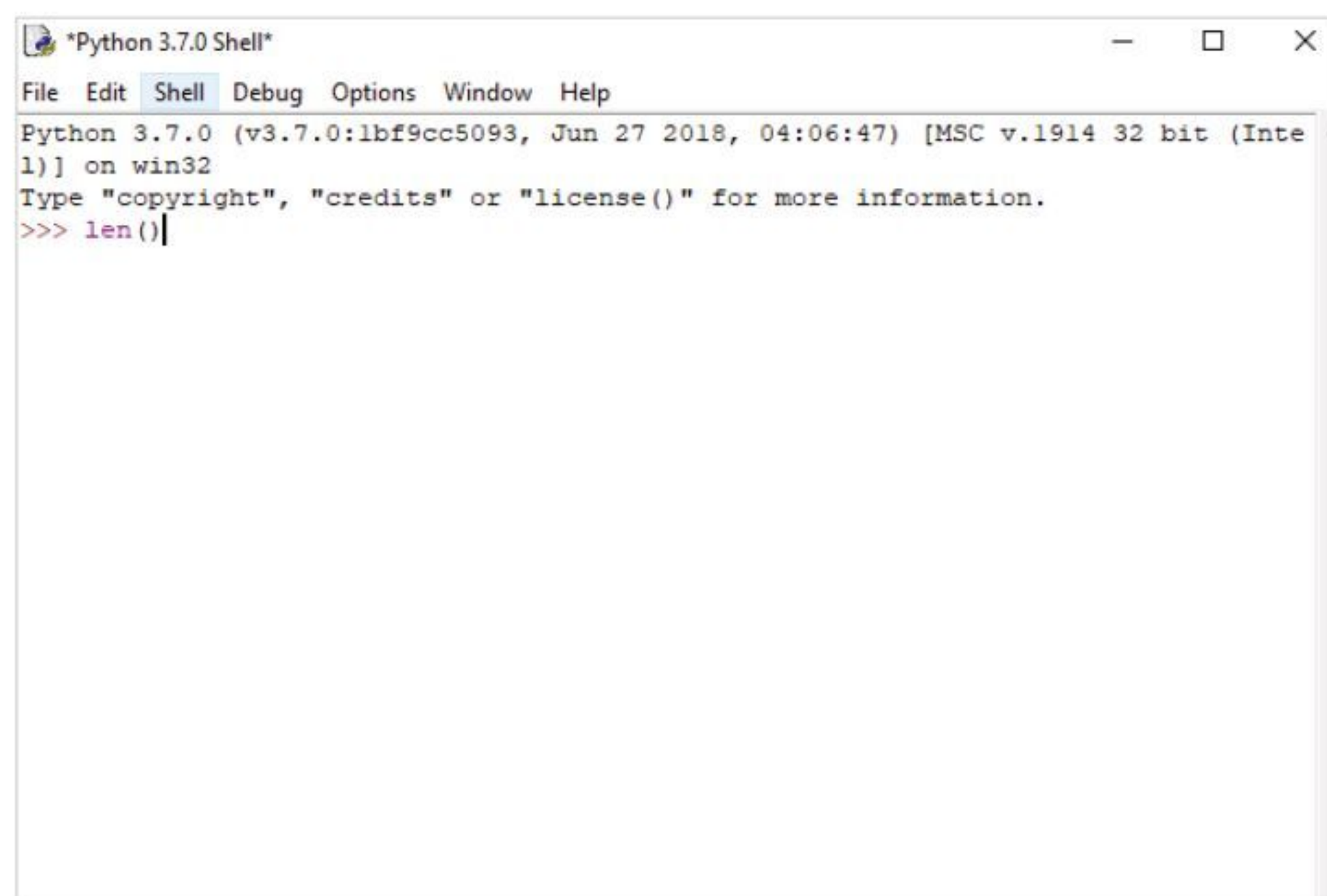
Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command, but Python enables you to define your own function.

FUNKY FUNCTIONS

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

STEP 1


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



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len()
>>>
```

STEP 2

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



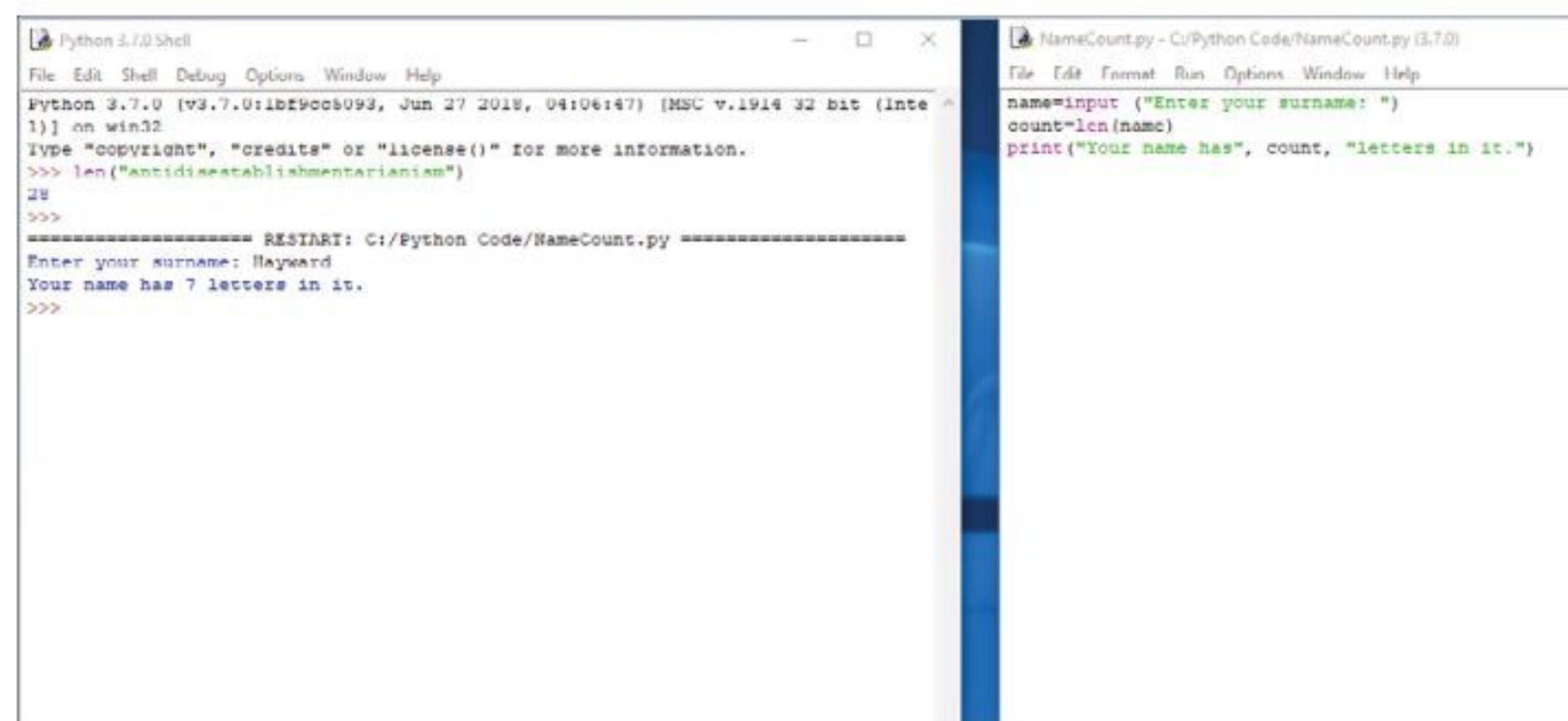
```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>
```

STEP 3

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

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

Press **F5** and save the code to execute it.

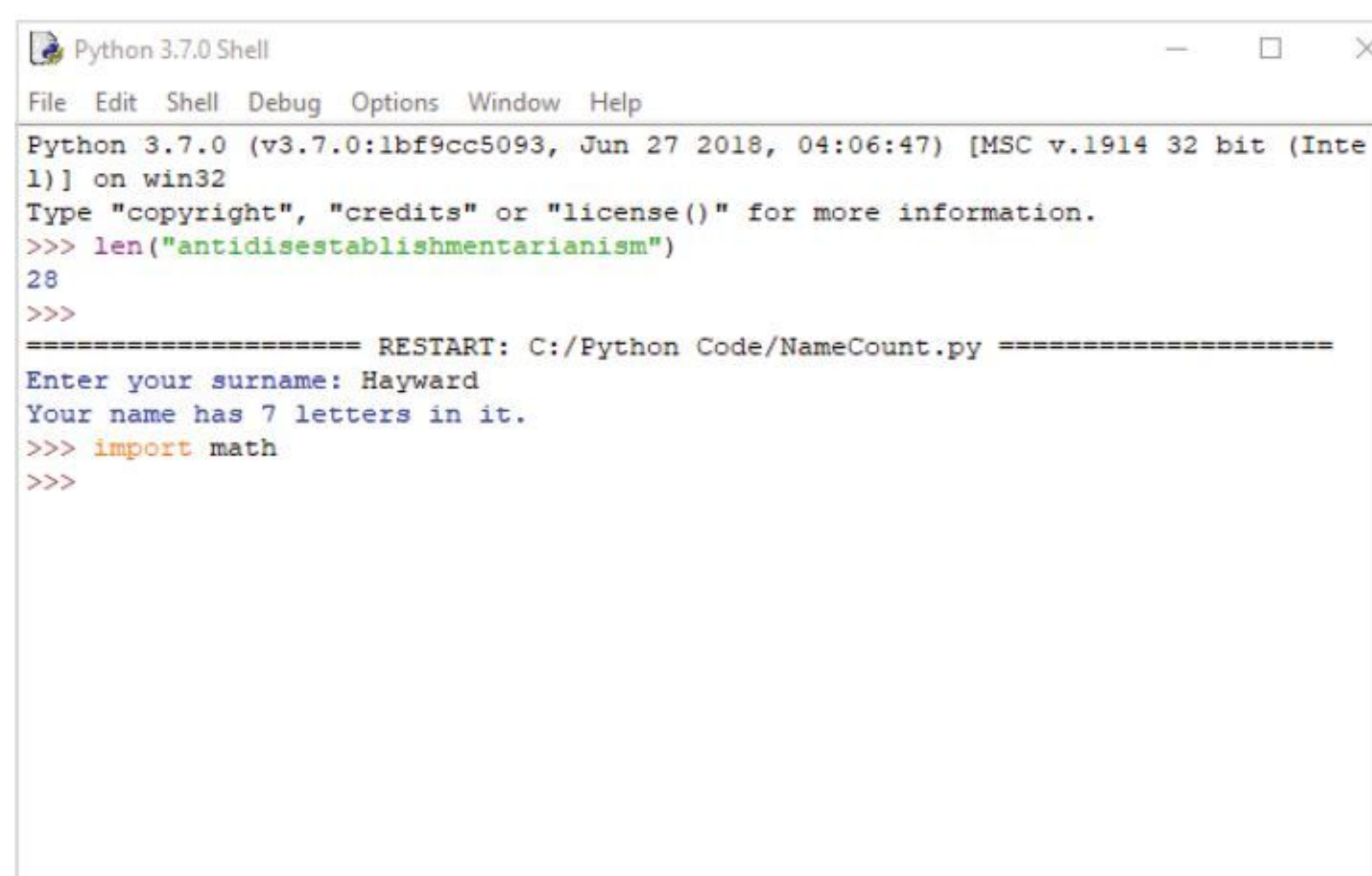


```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>

NameCount.py - C:/Python Code/NameCount.py (3.7.0)
File Edit Format Run Options Window Help
name=input("Enter your surname: ")
count=len(name)
print("Your name has", count, "letters in it.")
Enter your surname: Hayward
Your name has 7 letters in it.
>>>
```

STEP 4

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



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>

NameCount.py - C:/Python Code/NameCount.py (3.7.0)
File Edit Format Run Options Window Help
name=input("Enter your surname: ")
count=len(name)
print("Your name has", count, "letters in it.")
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

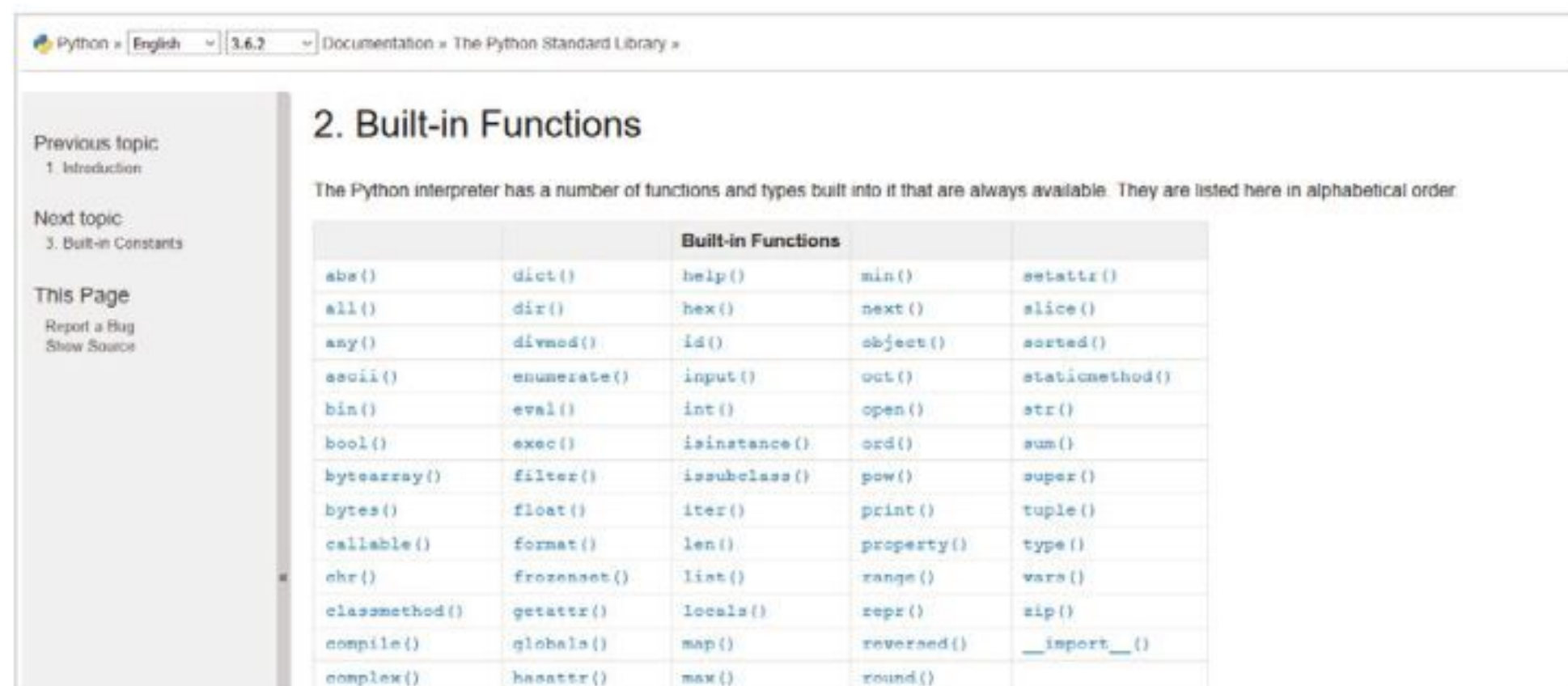



STEP 5

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

```
import math
```

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

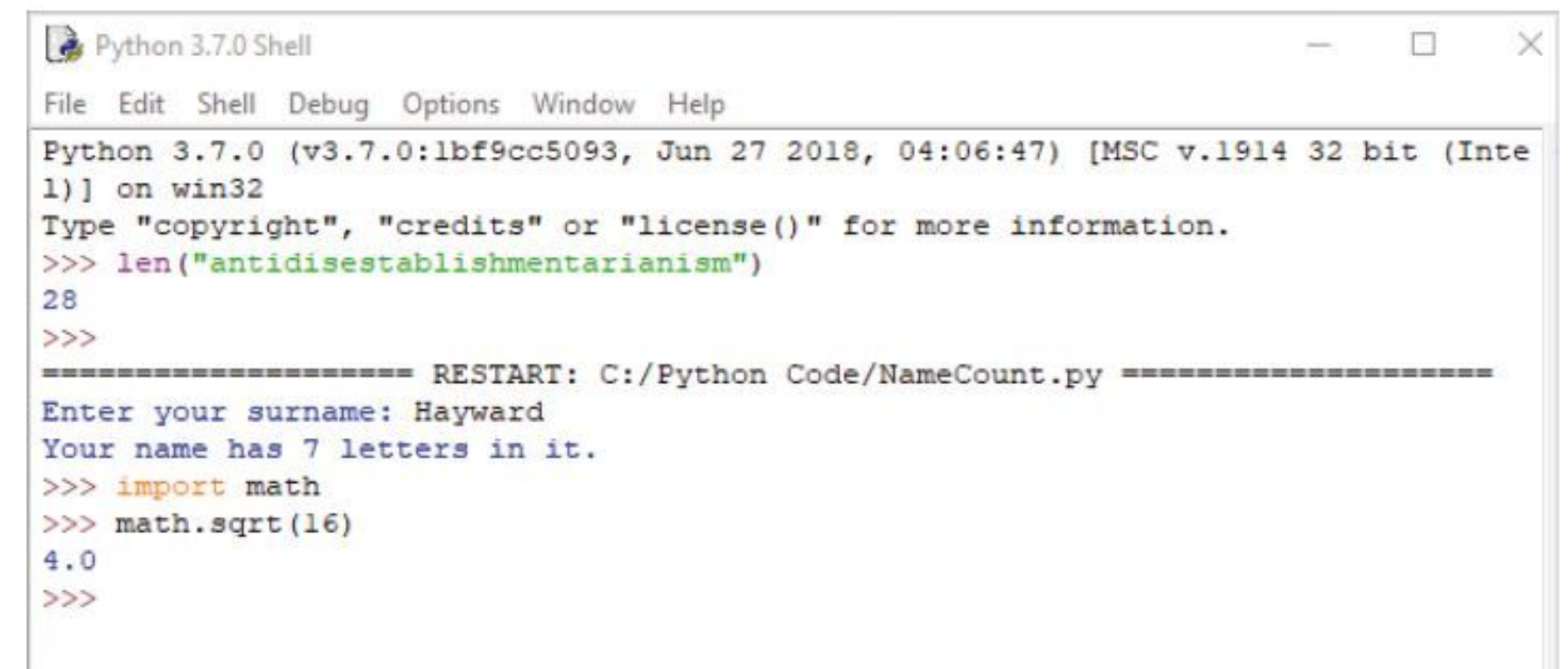


STEP 6

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

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

As you can see, the code is presented as **module.function(data)**.



FORGING FUNCTIONS

There are many different functions, created by other Python programmers, which you can import and you'll undoubtedly come across some excellent examples in the future. However, you can also create your own with the **def** command.

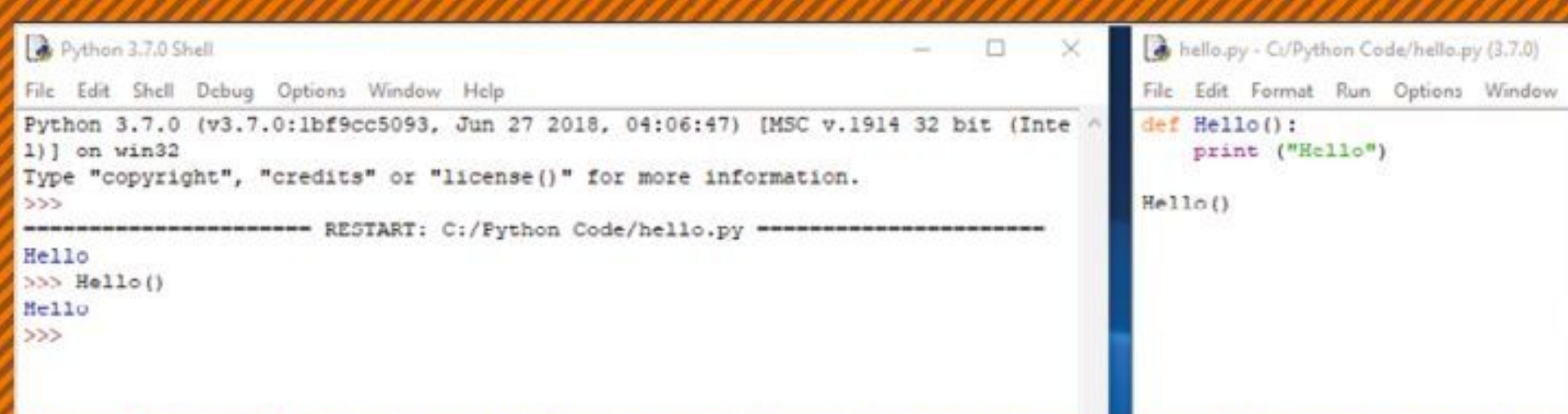
STEP 1

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

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

```
Hello()
```

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



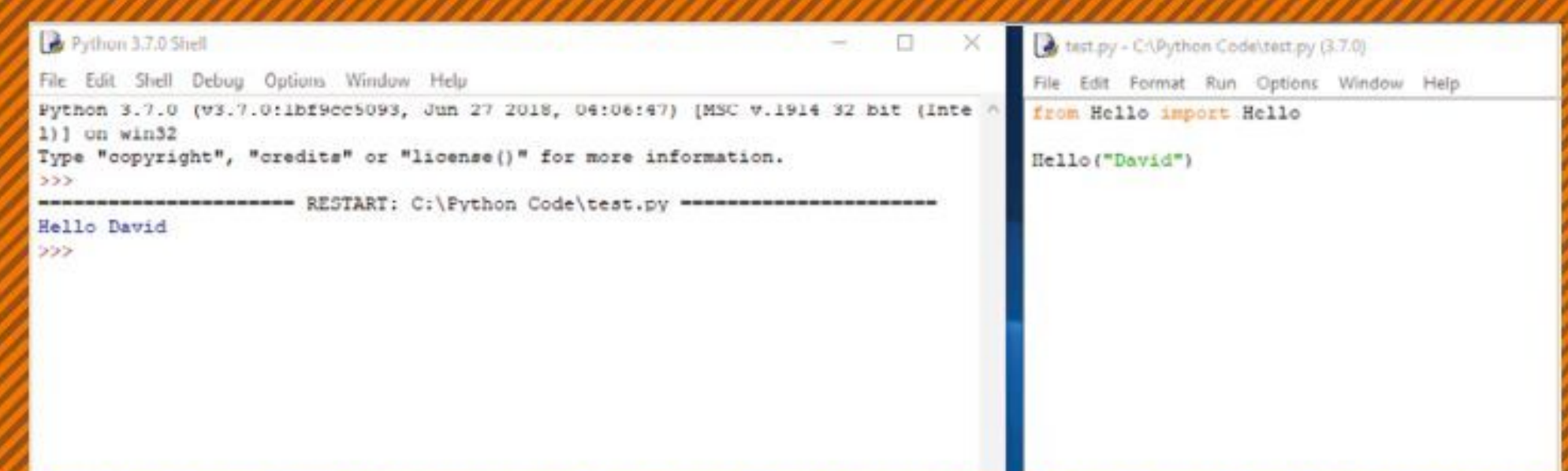
STEP 3

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

```
from Hello import Hello
```

```
Hello("David")
```

Press F5 to save and execute the code.



STEP 2

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

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

```
Hello("David")
```

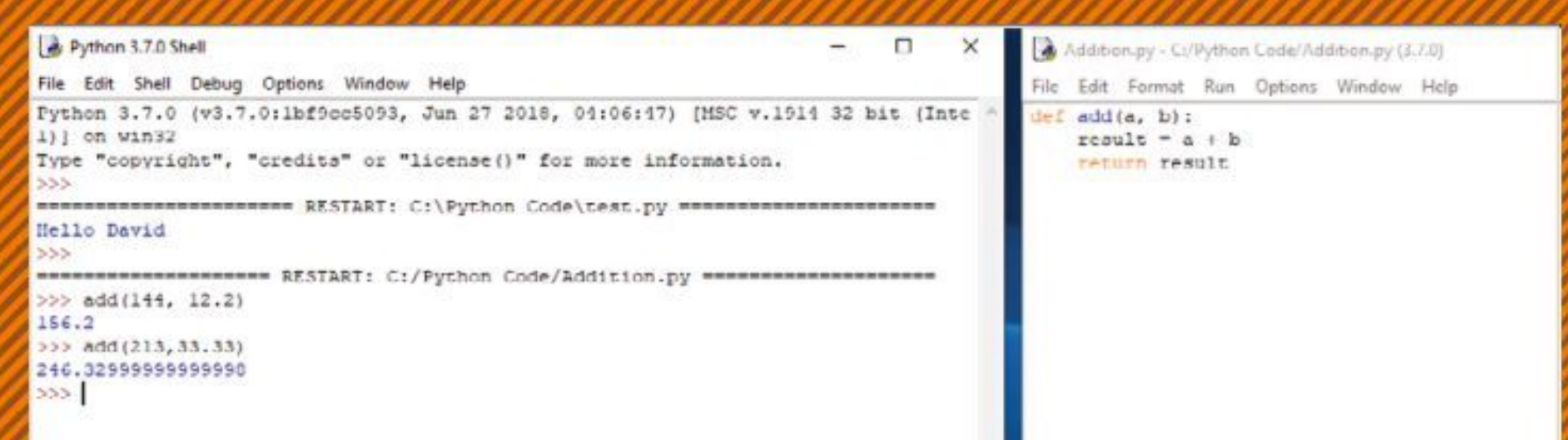
This will now accept the variable name, otherwise it will print Hello David. In the Shell, enter: `name= ("Bob")`, then, `Hello(name)`. Your function can now pass variables through it.



STEP 4

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

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





Conditions and Loops

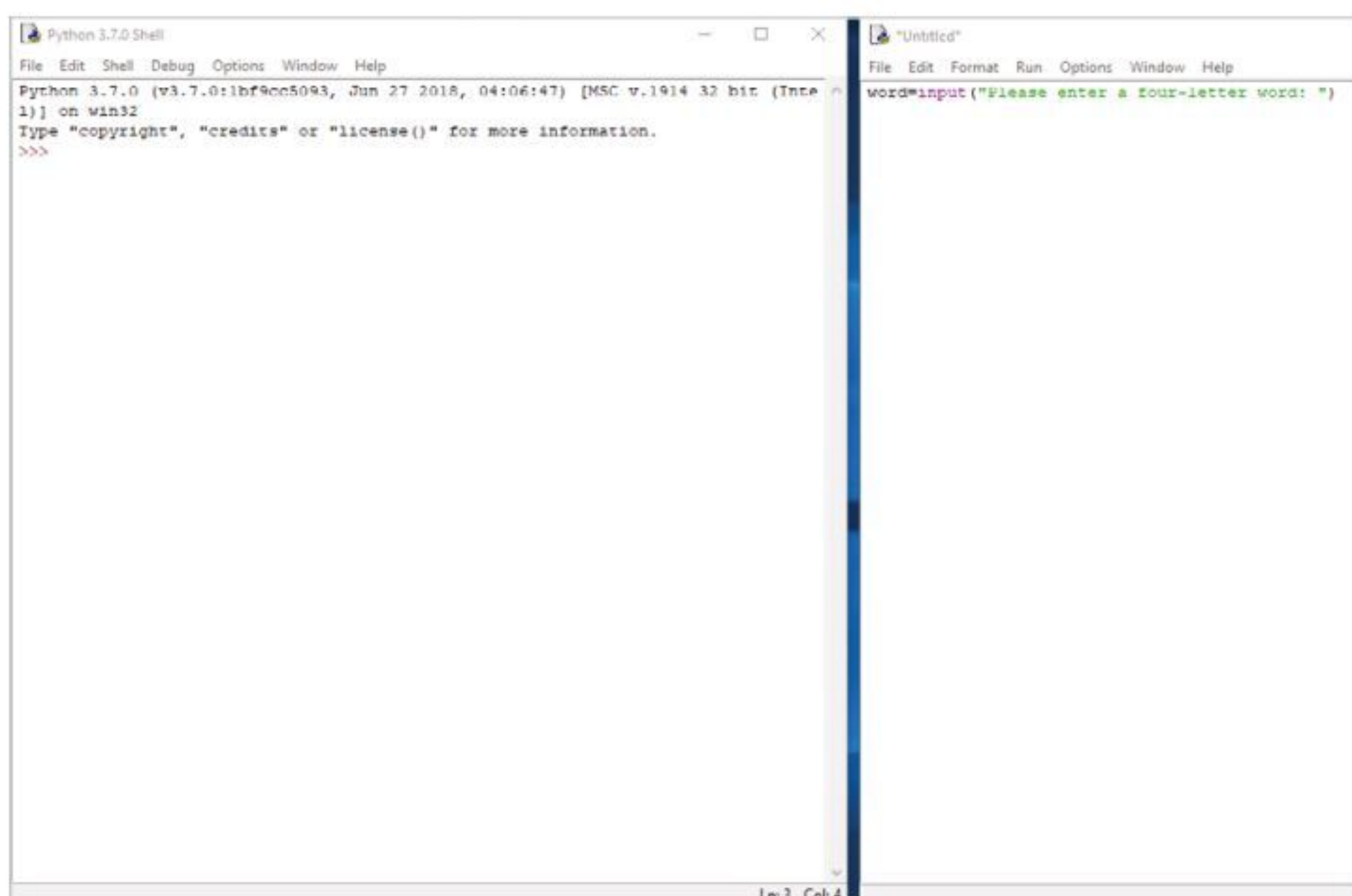
Conditions and loops are what make a program interesting, they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve, they could be the number of lives left in a game, or just displaying a countdown.

TRUE CONDITIONS

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

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

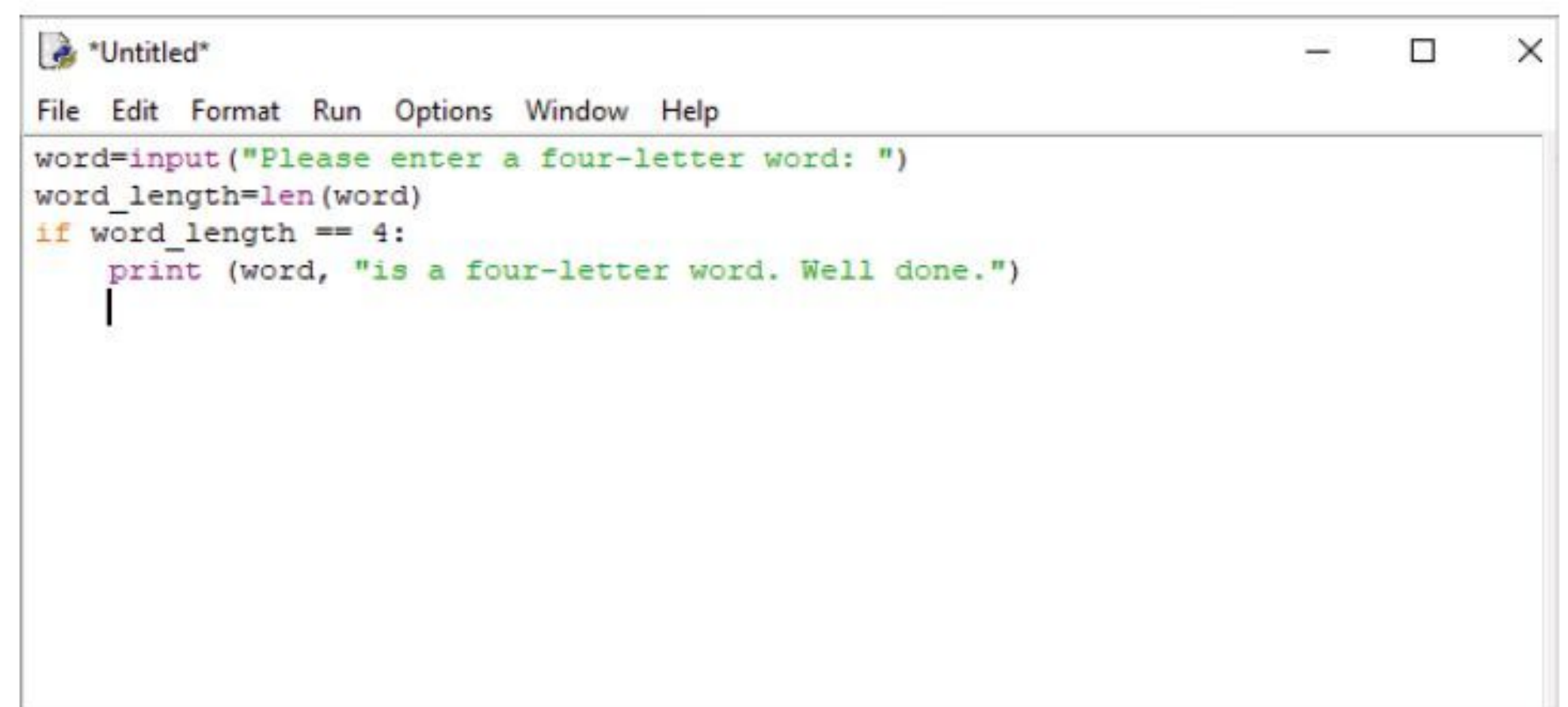
```
word=input("Please enter a four-letter word: ")
```



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

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

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



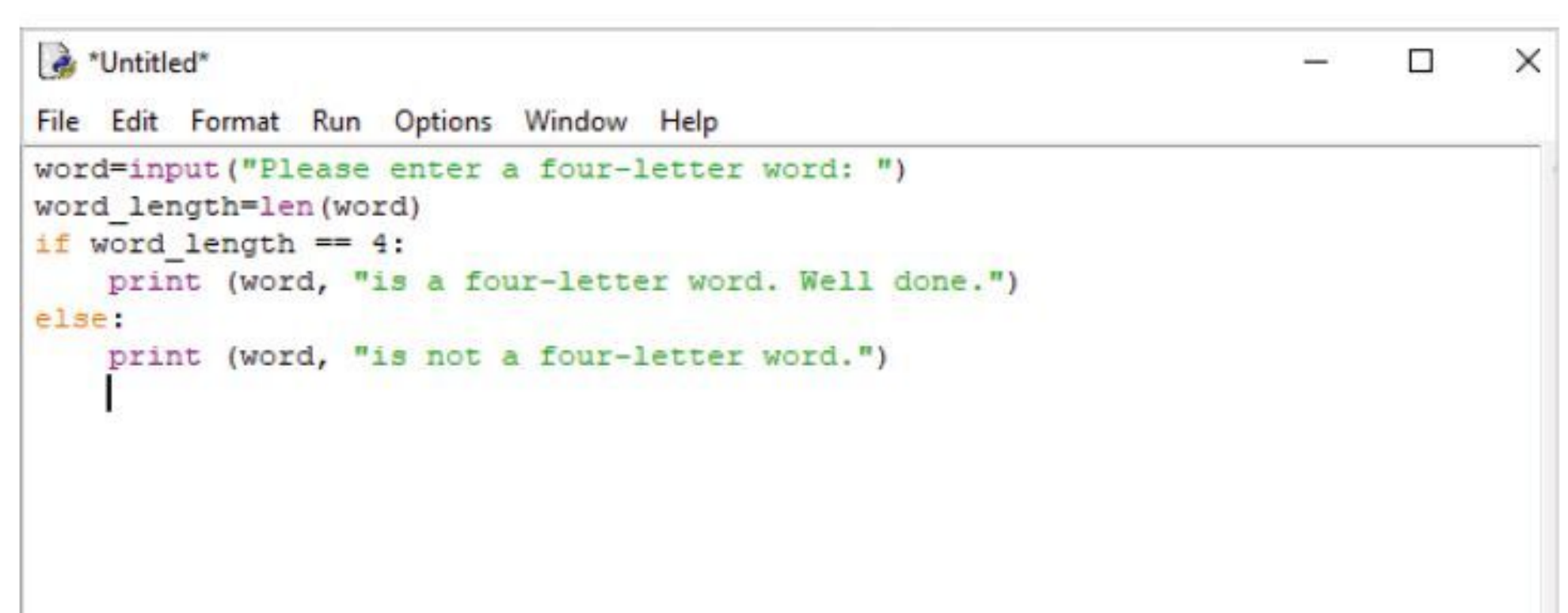
STEP 2 Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

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



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

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



STEP 5

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/wordgame.py =====
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>>
===== RESTART: C:/Python Code/wordgame.py =====
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>>

wordgame.py - C:/Python Code/wordgame.py (3.7.0)
File Edit Format Run Options Window 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.")
elif word_length == 3:
    print(word, "is a three-letter word. Try again.")
else:
    print(word, "is not a four-letter word.")
```

STEP 6

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/wordgame.py =====
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>>
===== RESTART: C:/Python Code/wordgame.py =====
Please enter a four-letter word: Egg
Egg is a three-letter word. Try again.
>>>

wordgame.py - C:/Python Code/wordgame.py (3.7.0)
File Edit Format Run Options Window 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.")
elif word_length == 3:
    print(word, "is a three-letter word. Try again.")
else:
    print(word, "is not a four-letter word.")
```

LOOPS

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

STEP 1

Let's start with a simple while statement. Like if, this will check to see if something is TRUE, then run the indented code:

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

```
Untitled
File Edit Format Run Options Window Help
x=1
while x<10:
    print(x)
    x=x+1
```

STEP 3

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

```
words=["Cat", "Dog", "Unicorn"]
for word in words:
    print(word)
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/loop1.py =====
1
2
3
4
5
6
7
8
9
Cat
Dog
Unicorn
>>>

loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help
words=["Cat", "Dog", "Unicorn"]
for word in words:
    print(word)
```

STEP 2

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/loop1.py =====
1
2
3
4
5
6
7
8
9
>>>

loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help
x=1
while x<10:
    print(x)
    x=x+1
```

STEP 4

The for loop can also be used in the countdown example by using the range function:

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

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Python Code/loop1.py =====
1
2
3
4
5
6
7
8
9
Cat
Dog
Unicorn
>>>

loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help
for x in range(1, 10):
    print(x)
```




Python Modules

We've mentioned modules previously, using the Math module as an example, but since using modules is such a large part of getting the most from Python it's worth dedicating a little more time to them.

MASTERING MODULES

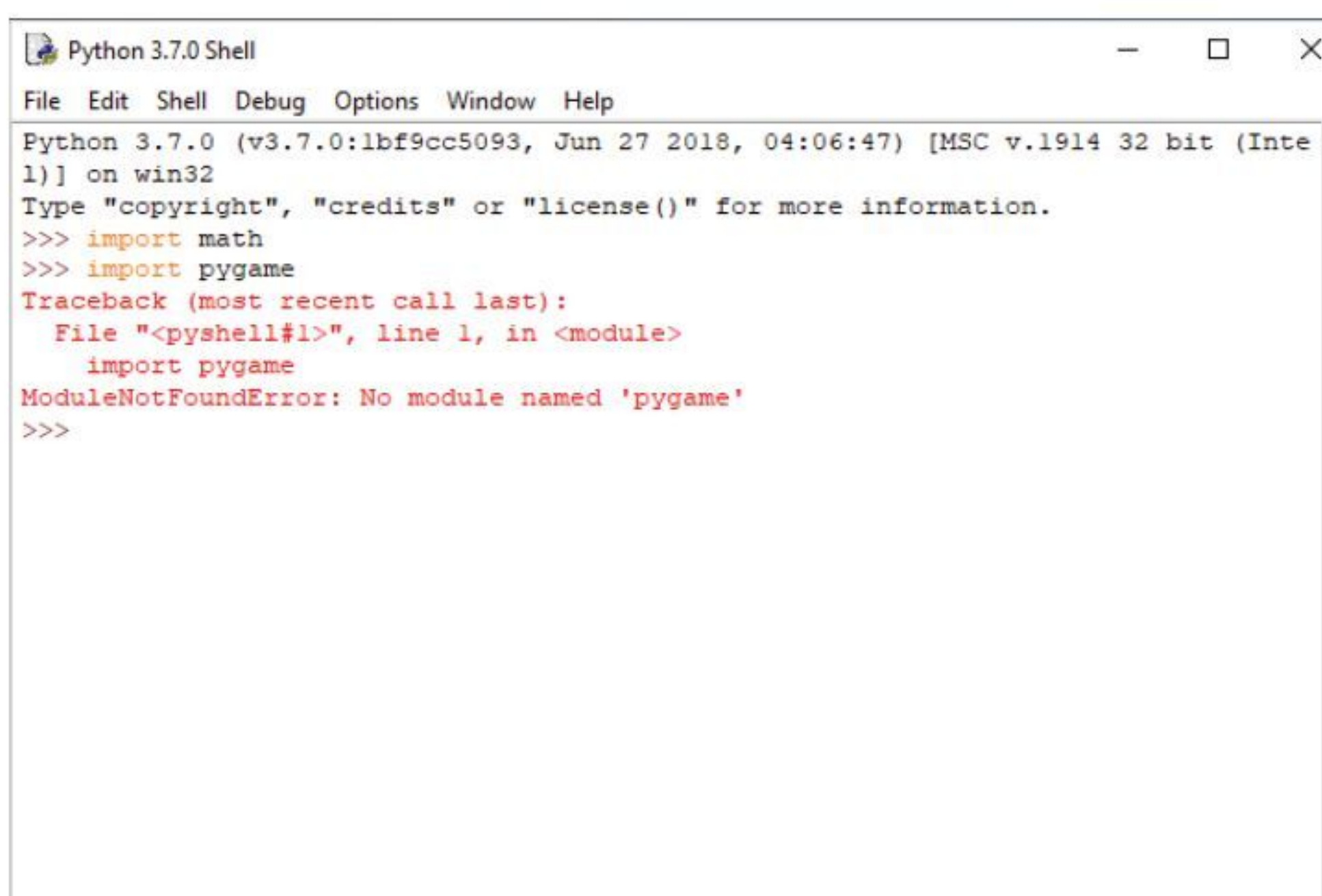
Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available, and as we've seen, you can even make your own.

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



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> |
```

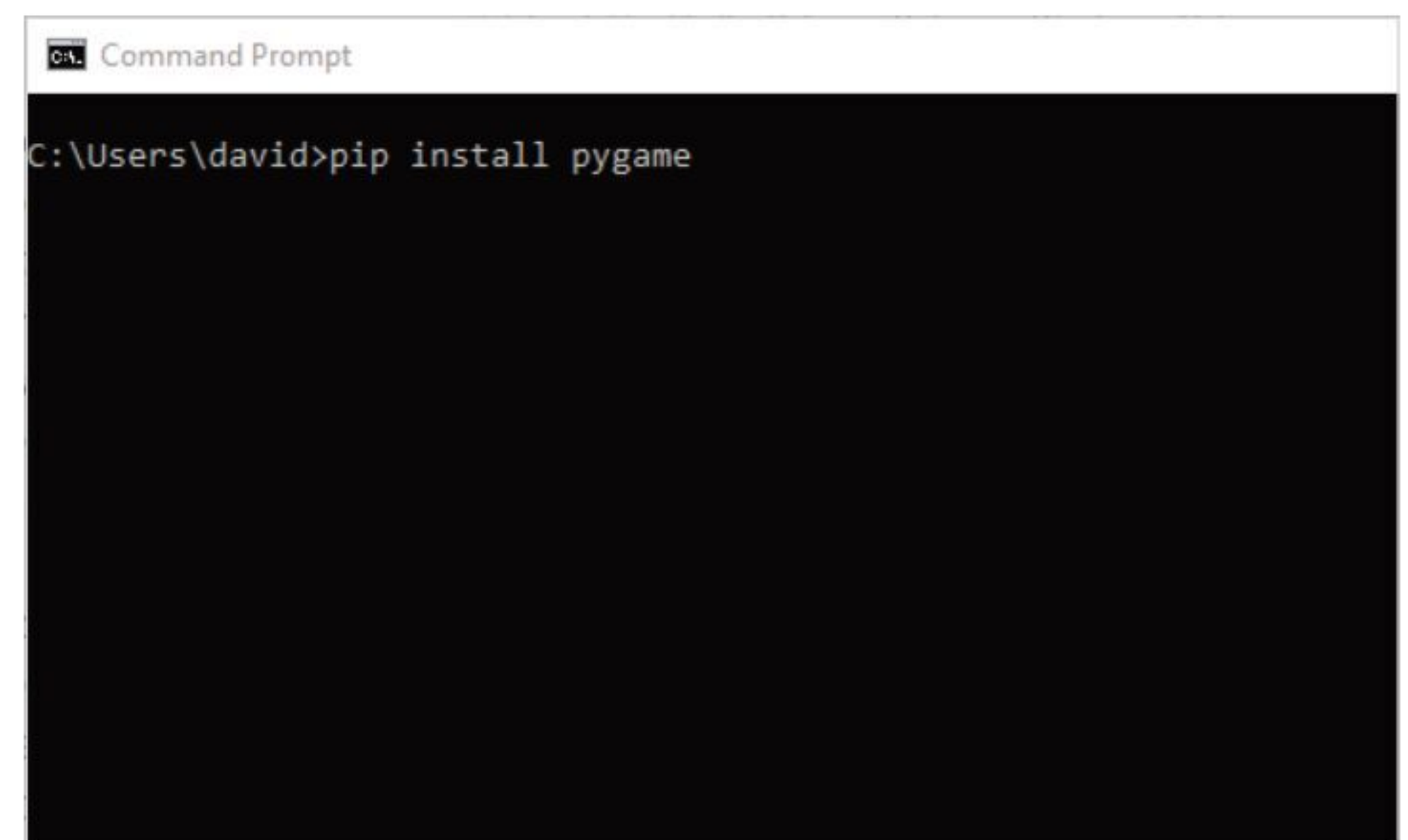
STEP 2 Some modules, especially on the Raspberry Pi, are included by default; the Math module is a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the Pygame module, which contains many functions to help create games. Try: `import pygame`.



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> import pygame
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
>>>
```

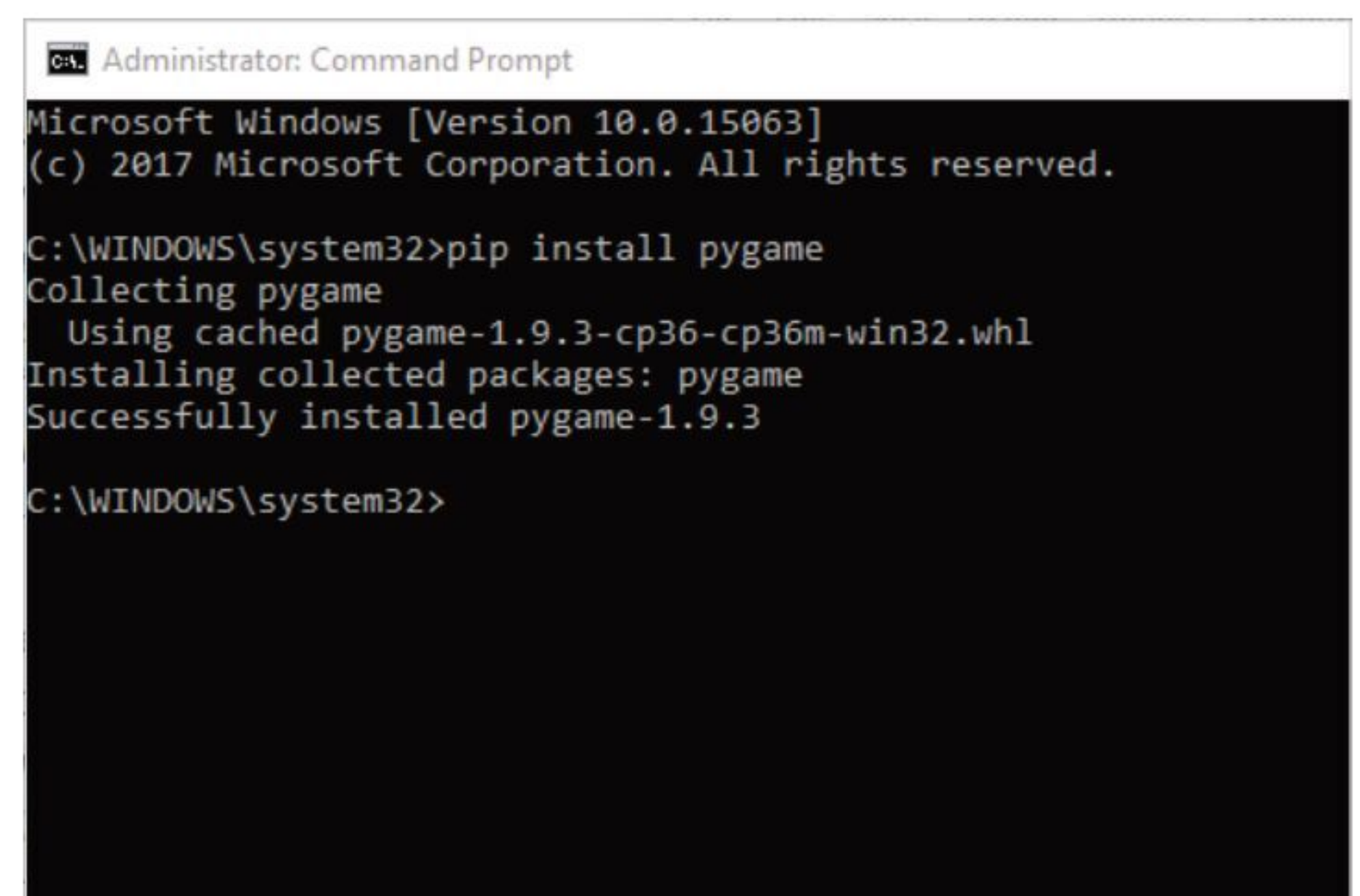
STEP 3 The result is an error in the IDLE Shell, as the Pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

```
pip install pygame
```



```
C:\Users\david>pip install pygame
```

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



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install pygame
Collecting pygame
  Using cached pygame-1.9.3-cp36-cp36m-win32.whl
Installing collected packages: pygame
Successfully installed pygame-1.9.3

C:\WINDOWS\system32>
```




STEP 5

Close the command prompt or Terminal, and re-launch the IDLE Shell. When you now enter `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded, or copied, from the Internet will contain a module, mainstream or unique, and their absence is commonly the source of errors in execution.

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
>>>
```

STEP 6

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

```
import random
```

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

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

```
*Untitled*
File Edit Format Run Options Window Help
import random

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

STEP 7

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

```
import random

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

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>>
----- RESTART: C:/Python Code/rnd number.py -----
10
4
10
18
19
3
20
12
8
10
>>>
----- RESTART: C:/Python Code/rnd number.py -----
16
40
45
7
53
56
20
54
89
70
11
```

STEP 8

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

```
import random
import math

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

print(math.pi)
```

```
*rnd number.py - C:/Python Code/rnd number.py (3.7.0)*
File Edit Format Run Options Window Help
import random
import math

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

print(math.pi)
```

STEP 9

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

```
from random import randint

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

```
*rnd number.py - C:/Python Code/rnd number.py (3.7.0)*
File Edit Format Run Options Window Help
from random import randint

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

STEP 10

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

```
import math as m

print(m.pi)
```

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

```
*rnd number.py - C:/Python Code/rnd number.py (3.7.0)*
File Edit Format Run Options Window Help
import math as m

print(m.pi)
```