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

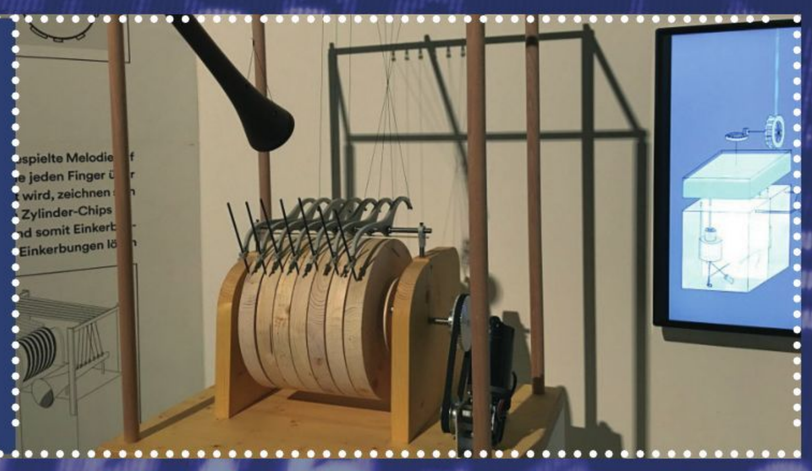
Essentially all forms of coding are made up of ones and zeros, on or off states. This works for a modern computer and even the oldest known computational device.

~87 BC

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.

~850 AD

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.



1800

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.

1842-1843

Ada Lovelace translated the memoirs of the Italian mathematician, Francis Maneclang, 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.

1930-1950

1951-1958

```
MONITOR FOR 6502 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

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

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

0013           RESETA  EQU     %00010011
0011           CTLREG  EQU     %00010001

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
               * DESTROYS: acc A
               * CALLS: none
               * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH     LDA A  ACIA       GET STATUS
C013 47                 ASR A             SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC    INCH       RECIEVE 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
C02A 2B 09             BMI    HEXERR     NOT HEX
C02C 81 46             CMP A  #'F
C02E 2E 05             BGT    HEXERR
C030 80 07             SUB A  #7         FIX A-F
C032 84 0F     HEXRTS  AND A  #$0F       CONVERT ASCII TO DIGIT
C034 39                RTS

C035 7E C0 AF  HEXERR  JMP    CTRL       RETURN TO CONTROL LOOP
```

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.

1959

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.

1960-1970

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
```

1970-1985

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.

1990s-Present Day

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.

Choosing a Programming Language

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

SQL

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

JAVASCRIPT

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

JAVA

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

C#

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

PYTHON

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

C++

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

RUBY

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

PERL

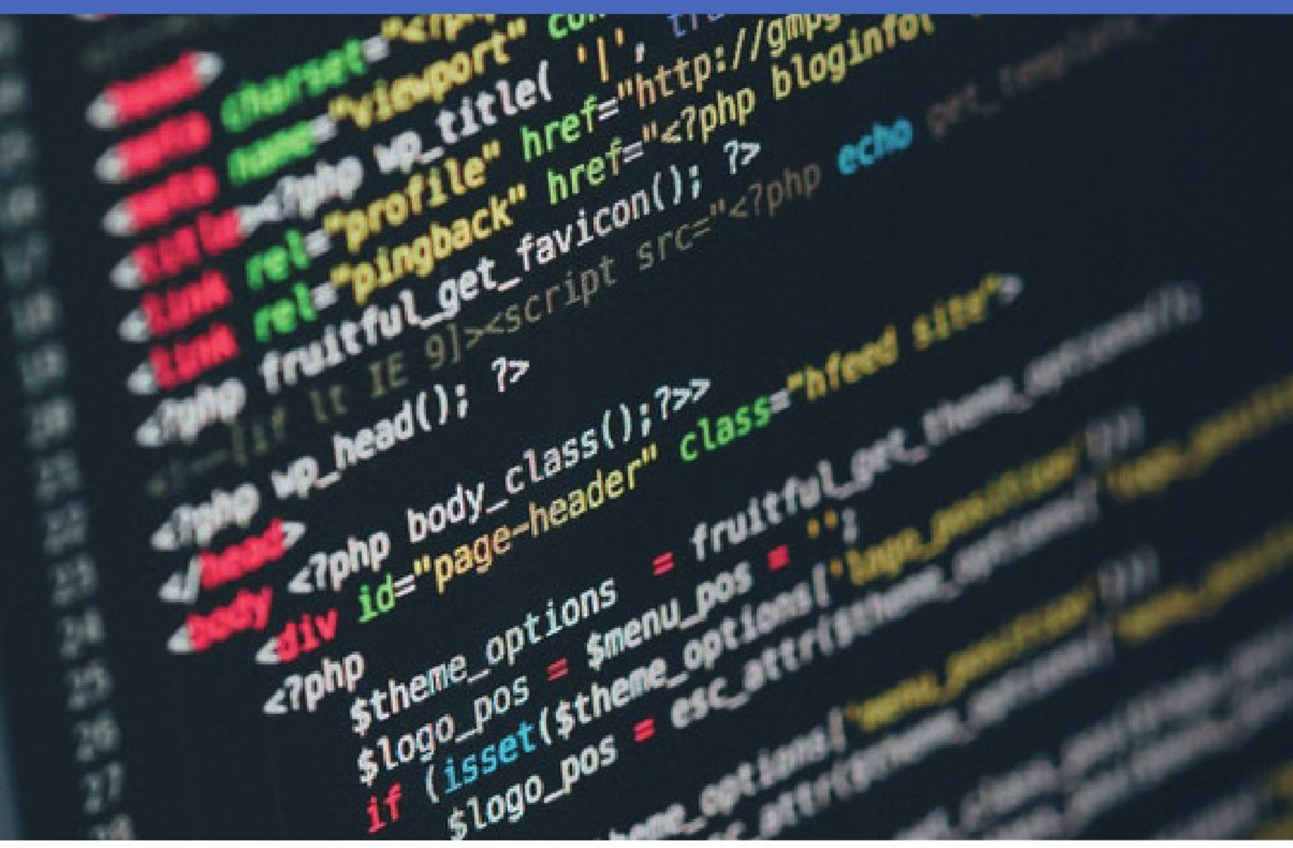
Perl is a general purpose programming language, used for a wide range of tasks including system administration, web development, network programming, GUI development and more. Its major features are that it’s easy to use, supports both procedural and object-oriented (OO) programming, has powerful built-in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

SWIFT

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

Creating a Coding Platform

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



Coding can be one of those experiences that sounds fantastic, but to get going with it, is often confusing. After all, there’s a plethora of languages to choose from, numerous 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. Then you access the Internet and 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 times.

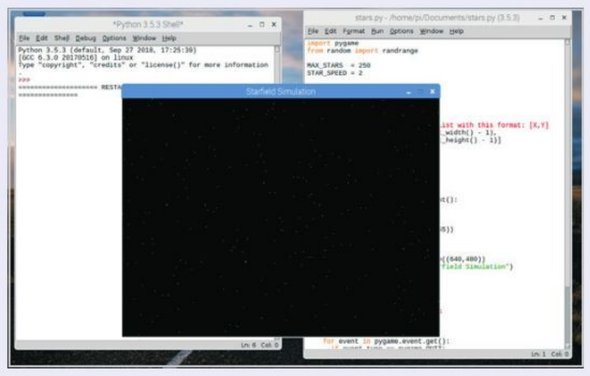
The trick is to slow down and, to begin with, not look too deeply into coding. Like all good projects, you need a solid foundation on which to build your skill and to have all the necessary tools available to hand to enable you to complete the basic steps. This is where creating a coding platform comes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

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 your 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.

Access to the Internet is necessary to download, install and update the coding development environment, alongside a computer with either: 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 - the tools that allow you to code, compile the code and execute it - are 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 changing from the mainstream development environment and using a collection of your own, discovered, tools to write your code in. 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 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.

Linux, however, is by far one of the most popular and important, coding operating systems available. Not only does it have a Unix-like backbone, but also 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 and it 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 that comes with its own customised Linux-based operating system, pre-installed with everything you need to start learning how to code in Python, C++, Scratch and more.

It's incredibly cheap, costing around £35 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.

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.

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, 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, for example, rather than having a single server with an installation of Windows Server, 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.

MINIX NEO N42C-4

The NEO N42C-4 is an extraordinarily small computer from mini-PC developer, MINIX. Measuring just 139 x 139 x 30mm, this Intel N4200 CPU powered, Windows 10 Pro pre-installed computer is one of the best coding platforms we've come across.

The beauty, of course, lies in the fact that with increased storage and memory available, you're able to create a computer that can easily host multiple virtual machines. The virtual machines can cover Linux, Android and other operating systems, allowing you to write and test cross-platform code without fear of damaging, or causing problems, with other production or home computers.

The MINIX NEO N42C-4 starts at around £250, with the base 32GB eMMC and 4GB of memory. You'll need to add another hundred and fifty, or so, to increase the specifications, but consider that a license for Windows 10 Pro alone costs £219 from the Microsoft Store and you can begin to see the benefits of opting for a more impressive hardware foundation over the likes of the Raspberry Pi.

Using Virtual Machines

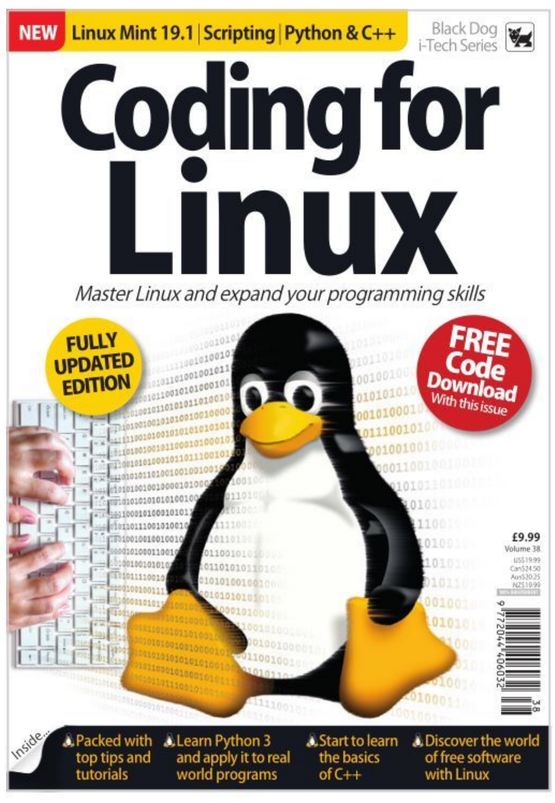
A Virtual Machine allows you to run an entire operating system from within an app on your desktop. This way, you're able to host multiple systems in a secure, safe and isolated environment. In short, it's an ideal way to code.

Sounds good, but what exactly is a Virtual Machine and how does it work?

The official definition of a virtual machine is 'an efficient, isolated duplicate of a real computer machine'. This basically means that a virtual machine is an emulated computer system that can operate in exactly the same way as a physical machine, but within the confines of a dedicated virtual machine operator, or Hypervisor.

The Hypervisor itself, is an app that will allow you to install a separate operating system, creating a virtual computer system within itself complete with access to the Internet, your home network and so on.

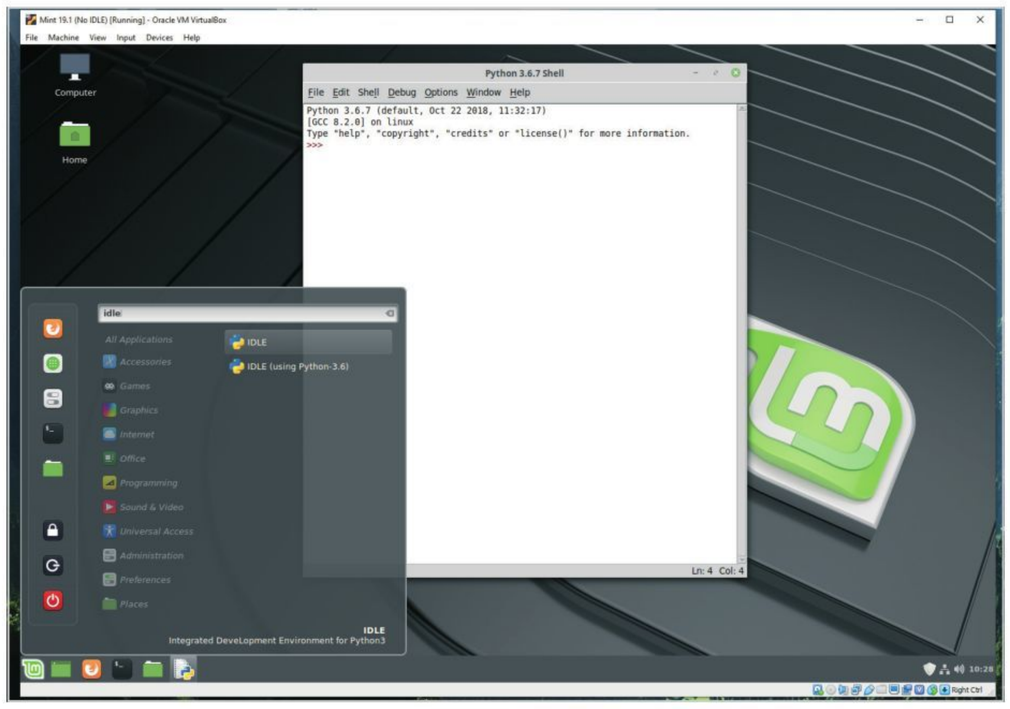
The Hypervisor will take resources from the host system - your physical computer, to create the virtual computer. This means that part of your physical computer's: memory, CPU, hard drive space and other shared resources, will be set aside for use in the virtual machine and therefore won't be available to the physical computer until the hypervisor has been closed down.



Our Linux titles contain steps on how to install a hypervisor and OS.

This resource overhead can be crippling for the physical machine if you don't already have enough memory, or hard drive space available, or your computer has a particularly slow processor. While it's entirely possible to run virtual machines on as little as 2GB of memory, it's not advisable. Ideally, you will need a minimum of 8GB of memory (you can get away with 4GB, but again, your physical computer will begin to suffer with the loss of memory to the virtual machine), at least 25 to 50GB of free space on your hard drive and a quad-core processor (again, you can have a dual-core CPU, but that can cause a bottleneck on your physical computer).

The limit to how many different virtual machines you host on your physical computer is restricted, therefore, by the amount of physical system resources you can allocate to each, while still leaving enough for your physical computer to operate on.



You're able to install Linux, and code inside a virtual machine on a Windows 10 host.

VIRTUAL OS

From within a hypervisor you're able to run a number of different operating systems. The type of OS depends greatly on the hypervisor you're running, as some are better at emulating a particular system over others. For example, VirtualBox, a free and easy to use hypervisor from Oracle, is great at running Windows and Linux virtual machines, but isn't so good at Android or macOS. QEMU is good for emulating ARM processors, therefore ideal for Android and such, but it can be difficult to master.

There are plenty of hypervisors available to try for free, with an equal amount commercially available that are significantly more powerful and offer better features. However, for most users, both beginner and professional, VirtualBox does a good enough job.

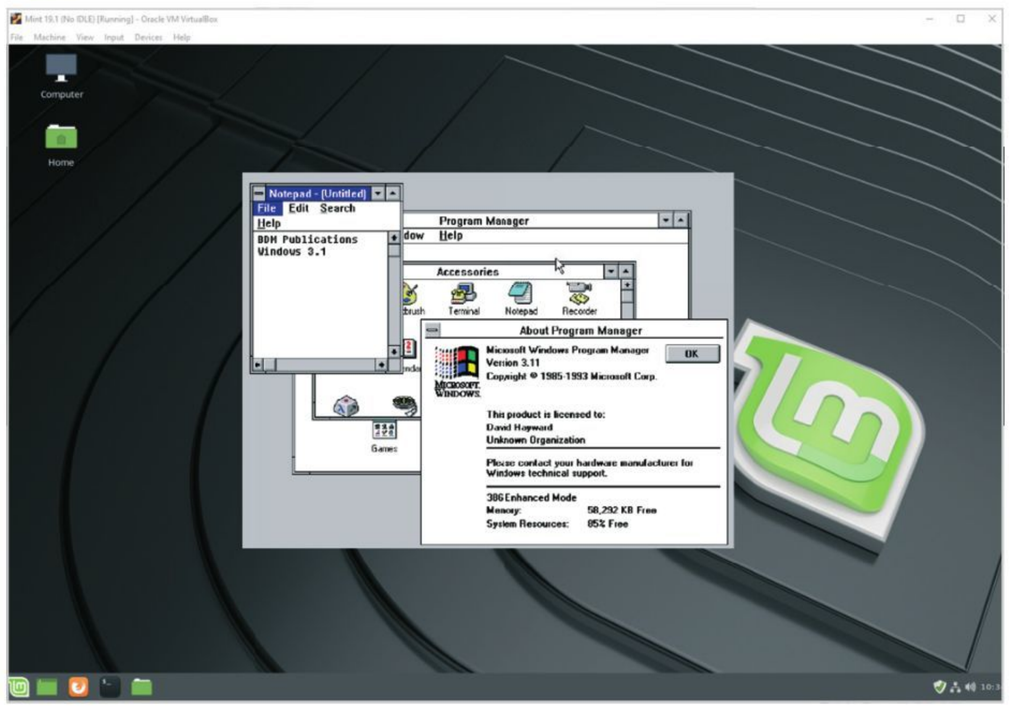
Within a hypervisor, you're able to set up and install any of the newer distributions of Linux, or if you feel the need, you're also able to install some of the more antiquated versions. You can install early versions of Windows, even as far back as Windows 3 complete with DOS 6.22 – although you may find some functionality of the VM lost due to the older drivers (such as access to the network).

With this in mind then, you're able to have an installation of Linux Mint, or the latest version of Ubuntu, running in an app on your Windows 10 PC. This is the beauty of using a virtual machine. Conversely, if your physical computer has Linux as its installed operating system, then with a hypervisor you're able to create a Windows 10 virtual machine – although you will need to have a licence code available to register and activate Windows 10.

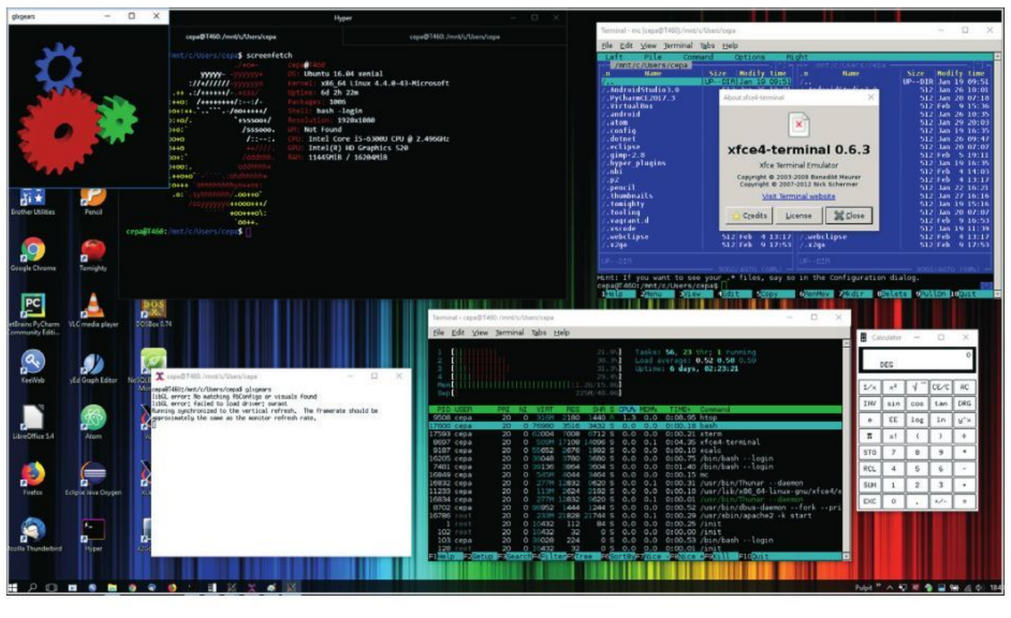
Using virtual machines removes the need to dual-boot. Dual-booting is having two, or more, physical operating systems installed on the same, or multiple, hard drives on a single computer. As the computer powers up, you're given the option to choose which OS you want to boot into. While this sounds like a more ideal scenario it isn't always as straight forward as it sounds, as all the operating systems that are booted into will have full access to the computer's entire system resources.

The problems with dual-booting come when one of the operating systems is updated. Most updates cover security patching, or bug fixing, however, some updates can alter the core - the kernel, of the OS. When these changes are applied, the update may alter the way in which the OS starts up, meaning the initial boot choice you made could be overwritten, leaving you without the ability to access the other operating systems installed on the computer. To rectify this, you'll need to access the Master Boot Record and alter the configuration to re-allow booting into the other systems. There's also the danger of possibly overwriting the first installed OS, or overwriting data and more often than not, most operating systems don't play well when running side-by-side. Indeed, while good, dual-booting has more than its fair share of problems. In contrast, using a virtual machine environment, while still problematic at times, takes out some of the more nasty and disastrous aspects of using multiple operating systems on a single computer.

Even old operating systems can be run inside a virtual machine.



Virtual machines can be as simple, or as complex as your needs require.

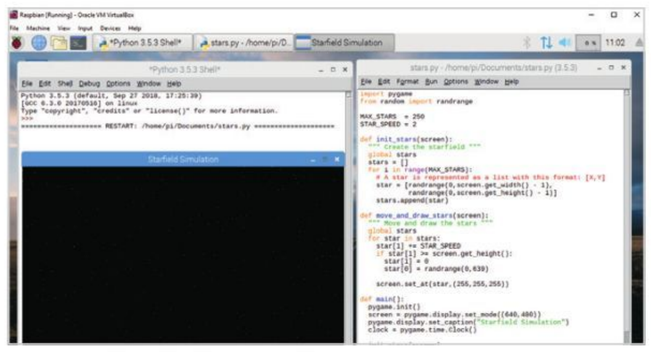


ADVANTAGES FOR CODERS

For the coder, having a virtual machine setup offers many advantages, the most popular being cross-platform code. Meaning if you write code within Windows 10, then with an installation of a Linux distro in a hypervisor, you're able to quickly and effortlessly power up the virtual machine and test your code in a completely different operating system. From this, you're able to iron out any bugs, tweak the code so it works better on a different platform and expand the reach of your code to non-Windows users.

The advantage of being able to configure a development environment, in specific ways for specific projects, is quite invaluable. Using a virtual machine setup greatly reduces the uncertainties that are inherent to having multiple versions of programming languages, libraries, IDEs and modules installed, to support the many different projects you may become involved in as a coder. Elements of code that 'talk' directly to specifics of an operating system can easily be overcome, without the need to clutter up your main, host system with cross-platform libraries, which in turn may have an affect on other libraries within the IDE.

Another element to consider is stability. If you're writing code that could potentially cause some instability to the core OS during its development phase, then executing and testing that code on a virtual machine makes more sense than testing it on your main computer; where having to repeatedly reboot, or reset something due to the code's instabilities, can become inefficient and just plain annoying.



Coding in Python on the Raspberry Pi Desktop OS inside a VM on Windows 10!

The virtual machine environment can be viewed as a sandbox, where you're able to test unsecure, or unstable code without it causing harm, or doing damage, to your main, working computer. Viruses and malware can be isolated within the VM without infecting the main computer, you're able to set up anonymity Internet use within the VM and you're able to install third-party software without it slowing down your main computer.

GOING VIRTUAL

While you're at the early stages of coding, using a virtual machine may seem a little excessive. However, it's worth looking into because coding in Linux can often be easier than coding in Windows, as some versions of Linux have IDEs pre-installed. Either way, virtualisation of an operating system is how many of the professional and successful coders and developers work, so getting used to it early on in your skill set is advantageous.

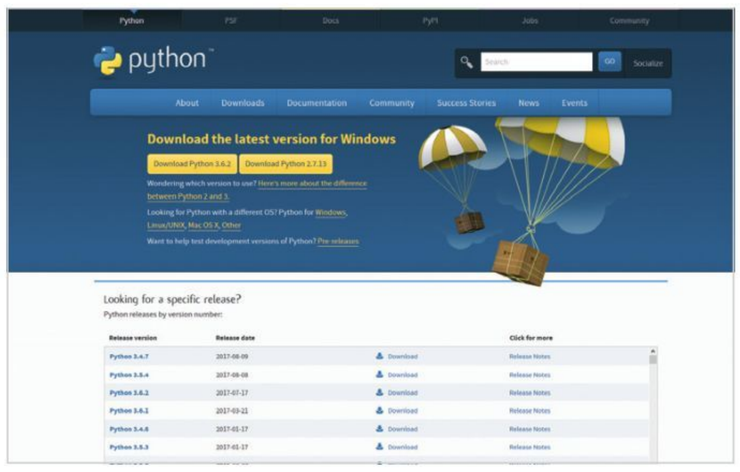
To start, look at installing VirtualBox. Then consider taking a look at our Linux titles, **https://bdmpublications.com/?s=linux&post_type=product**, to learn how to install Linux in a virtual environment and how best to utilise the operating system.

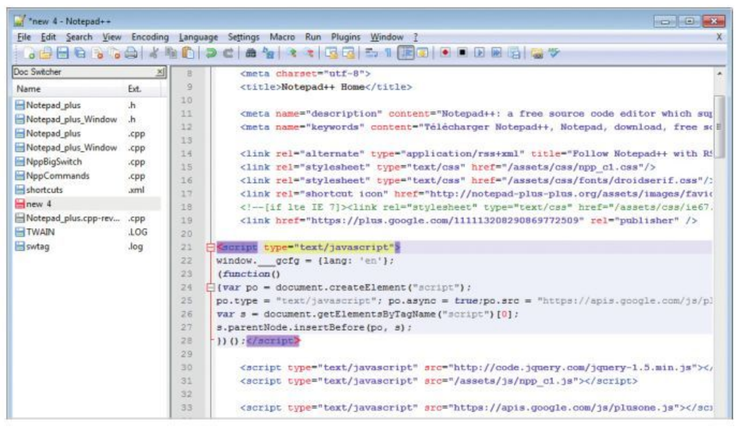
Equipment You Will Need

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

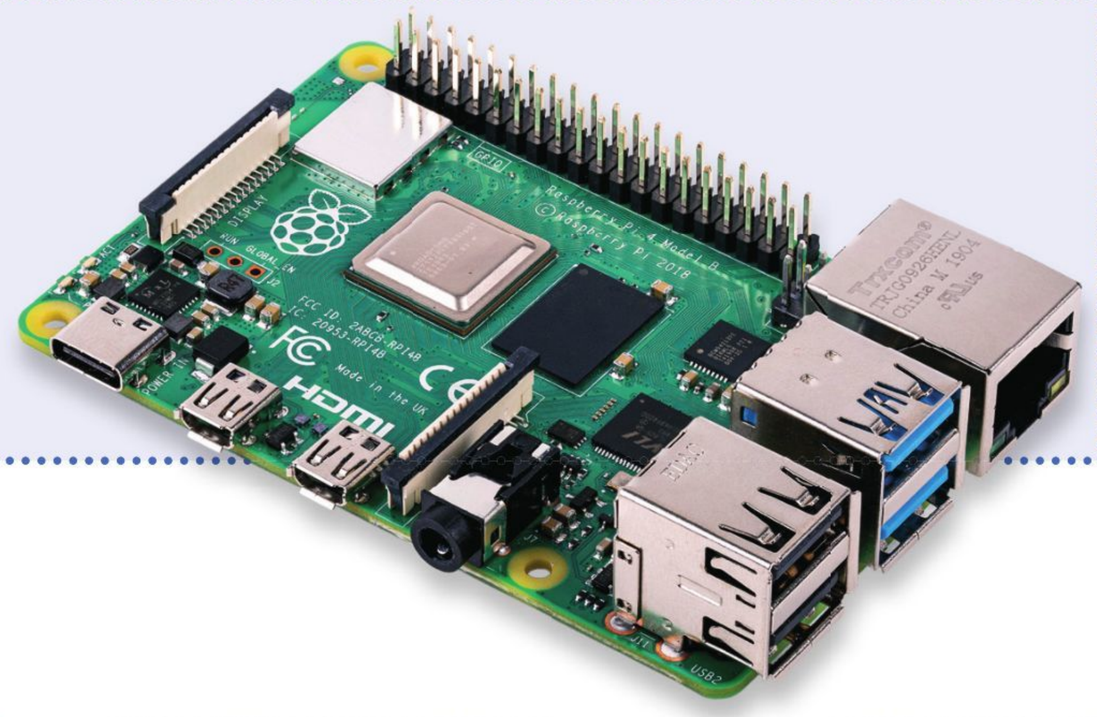
Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand you're learning something entirely new, and you will get there.

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 £33, 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.

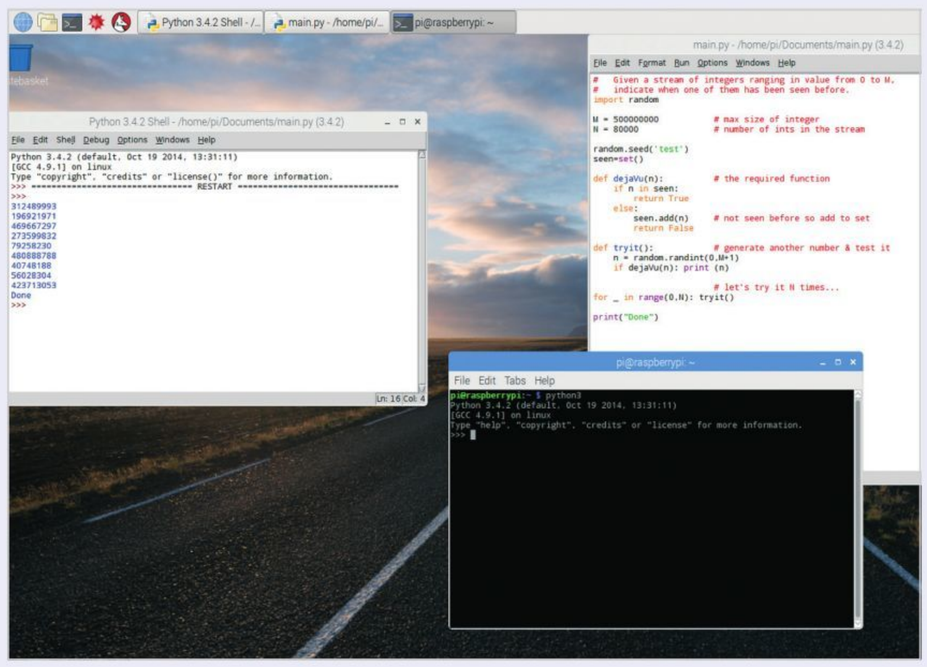


FUZE PROJECT

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.