Python Coding For Beginners

Get started with new programming skills

Over 450 Tips & Hints inside

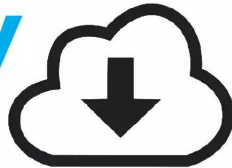
- Jargon-free Tips & Advice
- Step-by-step Tutorials
- Clear Full Colour Guides

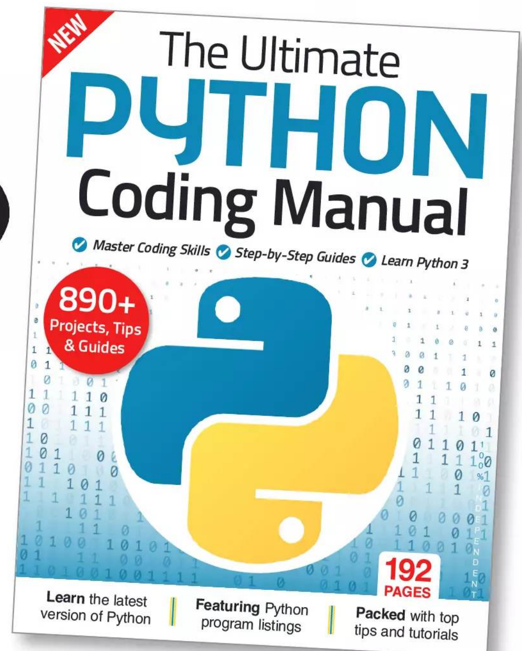
100% INDEPENDENT

Papercut

Get Your Exclusive FREE Gift Worth £9.99 Here!

Download Your FREE Copy of *The Ultimate Python Coding Manual*





Head over to your web browser and follow these simple instructions...

1/ Enter the following URL: **www.pclpublications.com/exclusives**

2/ Sign up/in and from the listings of our exclusive customer downloads, highlight the *The Ultimate Python Coding Manual* option.

3/ Enter your unique download code (Listed below) in the "Enter password to download" bar.

4/ Click the Download Now! Button and your digital manual will automatically download.

5/ Your file is a high resolution PDF file, which is compatible with the majority of customer devices/platforms.

Exclusive Download Code: **PCL64792CM**

NOTE: This is a limited reader offer running from 1st June 2024 to 1st September 2024.
Terms and conditions apply and can be found at www.pclpublications.com

Python

For Beginners

Python For Beginners is the first and only choice if you are a new adopter and want to learn everything you'll need to get started with coding. This independent guide is crammed with helpful guides and step-by-step fully illustrated tutorials, written in plain easy to follow English. Over the pages of this new user guide you will clearly learn all you need to know about coding your own amazing apps. With this unofficial instruction manual at your side no problem will be unsolvable, no question unanswered as you learn, explore and enhance your user experience.

Papercut

www.pclpublications.com

Contents

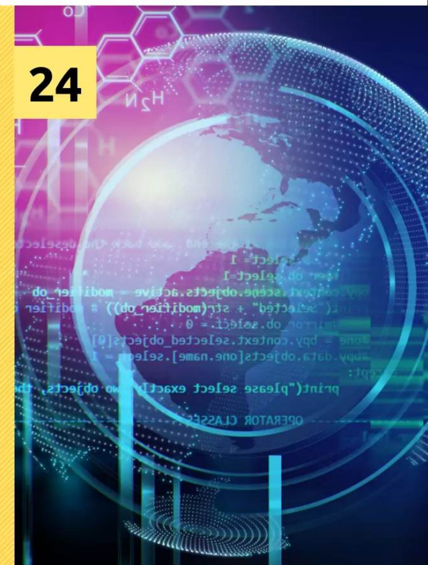
6 Getting Started

8 Being a Programmer
10 A Brief History of Coding
12 What can You Do with Python?
14 Python in Numbers
16 Why Python?
18 Python on the Pi
20 Using Virtual Machines
22 Creating a Coding Platform

Hello World

26 Equipment You Will Need
28 Getting to Know Python
30 How to Set Up Python in Windows
32 How to Set Up Python in Linux
34 Starting Python for the First Time
36 Your First Code
38 Saving and Executing Your Code
40 Executing Code from the Command Line
42 Numbers and Expressions
44 Using Comments
46 Working with Variables
48 User Input
50 Creating Functions
52 Conditions and Loops
54 Python Modules
56 Python Errors
58 Combining What You Know So Far
60 Python in Focus: Stitching Black Holes

24



62 Working with Data

64 Lists
66 Tuples
68 Dictionaries
70 Splitting and Joining Strings
72 Formatting Strings
74 Date and Time
76 Opening Files
78 Writing to Files
80 Exceptions
82 Python Graphics
84 Combining What You Know So Far
86 Python in Focus: Gaming

For more great Python, Linux & Raspberry Pi coding guides and tutorials **visit us at: www.pclpublications.com**

FREE CODE DOWNLOAD!

50 Complete programs!
Over 20,000 lines of code!

Visit PCL's Exclusive Code Portal; **www.pclpublications.com/exclusives**

Please note: Sign up required to access download file.

Getting Started

Python is a high-level, general-purpose programming language that was developed by Guido van Rossum in the late eighties and is based upon a number of other languages, while being the successor to the popular ABC language.

It has been devised to work on a human level, so it's readable and understandable without needing to delve into obscure volumes of machine code, hexadecimal characters, or even ones and zeros. It's clear, logical, comprehensive, powerful and functional, yet also easy to follow and learn.

You will find Python at the heart of some of the most interesting and cutting-edge technologies in the world. It's the code that binds supercomputer algorithms together; it's used in the space industry, and in science and engineering. AI, and the likes of Alexa and Siri, Cortana and the Google Assistant all utilise Python for their powerful voice recognition technology. It's simply an amazing, versatile and incredible language to learn.

So let's get started and explore what you need to become a Python programmer.

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.

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!

> “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 getup()
6   {
7       textcolor(BLACK);
8       textbackground(15);
9       clrscr();
10      window(10,2,70,3);
11      cprintf("Press X to Exit, Press Space to Jump");
12      window(62,2,80,3);
13      cprintf("SCORE : ");
14      window(1,25,80,25);
15      for(int x=0;x<79;x++)
16      cprintf("п");
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      cprintf("                 ");
32      cprintf("         мллллллм");
33      cprintf("         лллллллл");
34      cprintf("         лллллппп");
35      cprintf(" л      мллллппп ");
36      cprintf(" ллм  мллллллммм ");
37      cprintf(" плллллллллллл  п ");
38      cprintf("   пллллллллп     ");
39      if(jump==1 || jump==2){
40      cprintf("    ллп пл        ");
41      cprintf("    лм   лм       ");
42      }else if(a==1)
43      {
44      cprintf("    пллп  ппп     ");
45      cprintf("      лм          ");
46      a=2;
47      }
48      else if(a==2)
49      {
50      cprintf("     плм пл       ");
51      cprintf("          лм      ");
52      a=1;
53      }
54      cprintf("                ");
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.

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 01101001 01101110 01100111

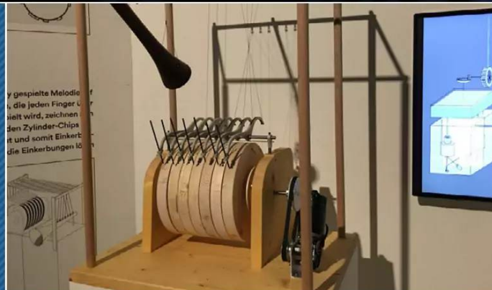
Essentially all forms of coding are made up of ones and zeros - on or off states. This works for both a modern computer and even the oldest known computational device.

~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

During the Second World War, 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

```
MONITOR FOR 6802 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 on which it was being used. 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

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, they could also freely share their code and source other code to improve their own.

What can You Do with Python?

Python is an open-source, object-oriented programming language that's simple to understand and write, 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 the ideal, first programmers' choice to its ability to create interactive stories and games; from scientific applications 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.

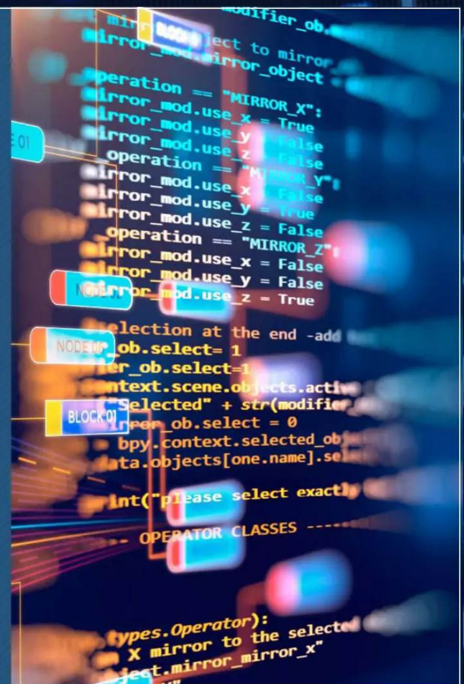
```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
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, in 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.



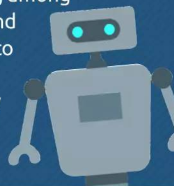
ARTIFICIAL INTELLIGENCE

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.

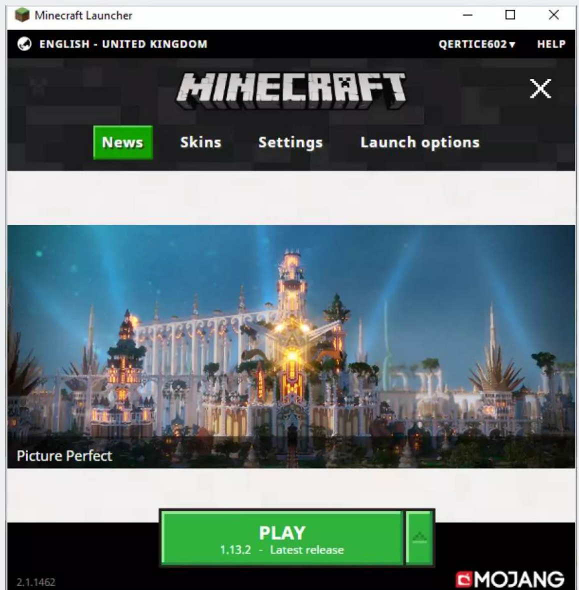
Spotify, for example, uses Python based code, among other things, to analyse your musical habits and offer playlists based on what you've listened to in the past. It's all clever stuff and, moving forward, Python is at the forefront of the way the Internet will work in the future.



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.

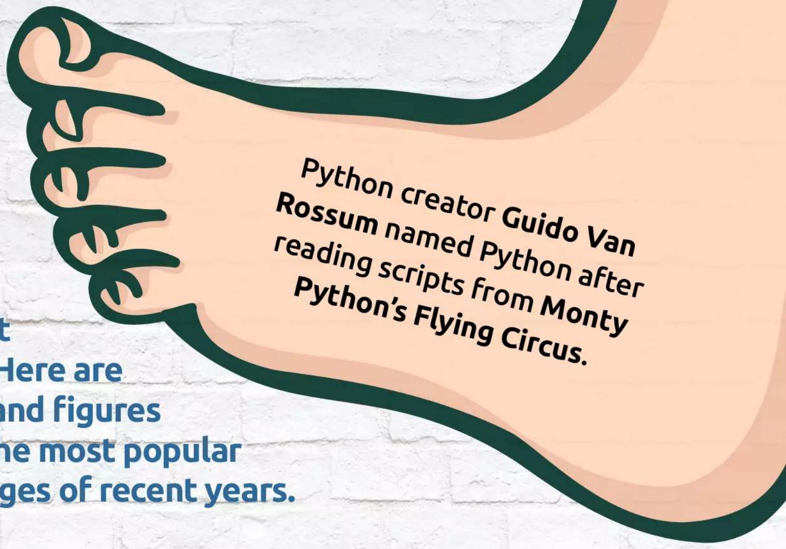
PYTHON EVERYWHERE

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.

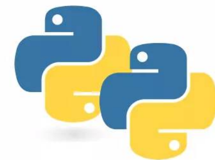
Whatever route you decide to take on your coding journey, you will do well to have Python in your corner.

PYTHON 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.



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

As of the end of 2023, Python was the most discussed language on the Internet.

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

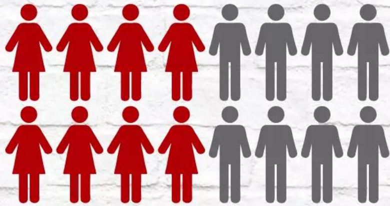
Disney Pixar uses Python in its Renderman software to operate between other graphics packages.

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%

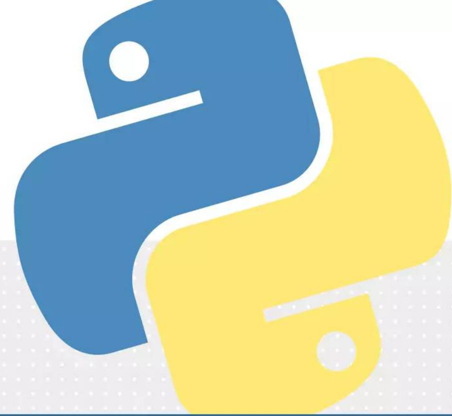
75% OF ALL PYTHON DEVELOPERS USE PYTHON 3, WHEREAS 25% STILL USE THE OUTDATED PYTHON 2 VERSION.

79%

79% OF ALL PROGRAMMERS USE PYTHON ON A DAY-TO-DAY BASIS, 21% USE IT AS A SECONDARY LANGUAGE.

49%

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



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

Bombs - GUI - TheIDE - [d:\uppsrc\CtrlLib\ArrayCtrl.cpp windows-1252] { examples }

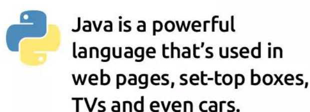
```
        SetCursor(p.y);
    Ctrl::ChildGotFocus();
}

void ArrayCtrl::ChildLostFocus()
{
    if(cursor >= 0)
        RefreshRow(cursor);
    Ctrl::ChildLostFocus();
}

void  ArrayCtrl::Paint(Draw& w) {
    LTIMING("Paint");
    Size size = GetSize();
    Rect r;
    r.bottom = 0;
    bool hasfocus = HasFocusDeep();
    int i = GetLineAt(sb);
    int xs = -header.GetScroll();
    int js;
    for(js = 0; js < column.GetCount(); js++) {
        int cw = header.GetTabWidth(js);
        if ( ( xs + cw - vertgrid + (js == column.GetCount() - 1)) >= 0)
            break;
        xs += cw;
    }
    Color fc = Blend(SColorDisabled, SColorPaper);
    if(!IsNull(i))
        while(i < GetCount()) {
            r.top = GetLineY(i) - sb;
            if(r.top > size.cy) break;
            r.bottom = r.top + GetLineCy(i);
            int x = xs;
            for(int j = js; j < column.GetCount(); j++) {
                int cw = header.GetTabWidth(j);
                int cm = column[j].margin;
                if(cm < 0)
                    cm = header.Tab(j).GetMargin();
                if(x > size.cx) break;
                r.left = x;
```

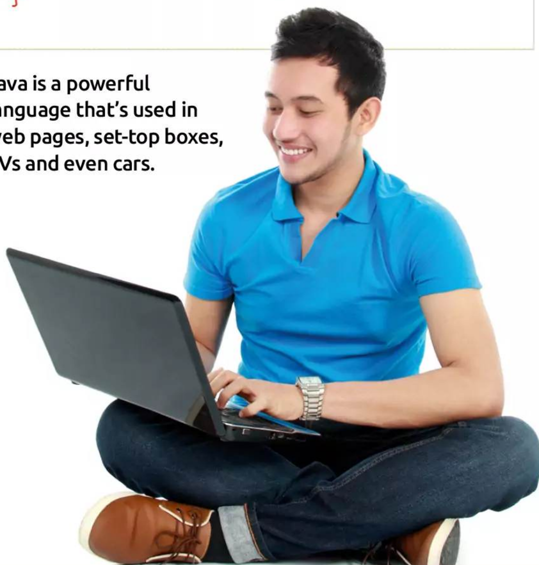
environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

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

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

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


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

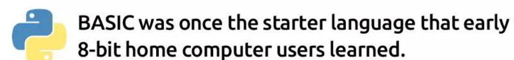


Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece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

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

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

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

0 OK, 0:1
```


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

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


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

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

    guessed_letters.append(player_guess)

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

    if player_guess not in chosen_word:
```

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

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.

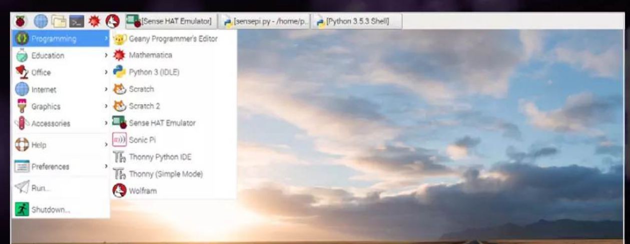
Using this method saves a lot of money, as you don't need another keyboard, mouse and monitor, plus, you won't need to allocate sufficient space to accommodate those extras either. If you're pushed for space and money, then for around £60, buying one of the many kits available will provide the Pi with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

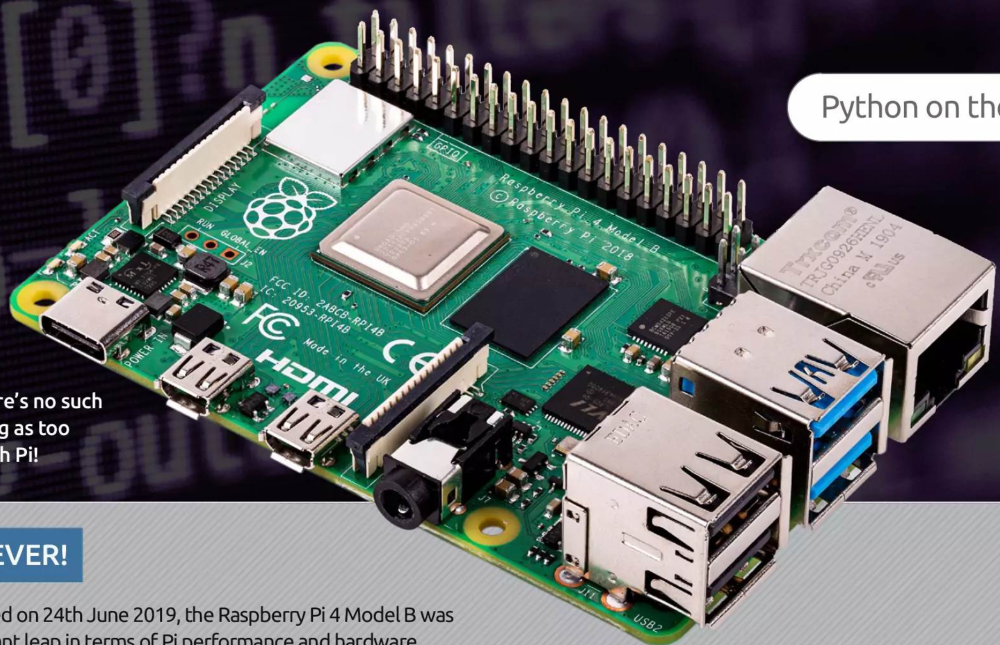
The main advantage is, of course, the extra content that the Raspberry Pi Foundation has included out of the box. The reason for this is that the Raspberry Pi's goal is to help educate the user, whether that's coding, electronics, or some other aspect of computing. To achieve that goal the Pi Foundation 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.

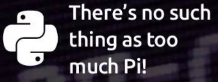
There are other advantages, such as being able to combine Python code with Scratch (an Object-Oriented programming language developed by MIT, for children to understand how coding works) and being able to code the GPIO connection on the Pi to further control any attached robotics or electronics projects. 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 was 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 1GB, 2GB, or 4GB memory versions, the Pi 4 is one-step 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 handing 4K resolutions and OpenGL ES 3.0 graphics.

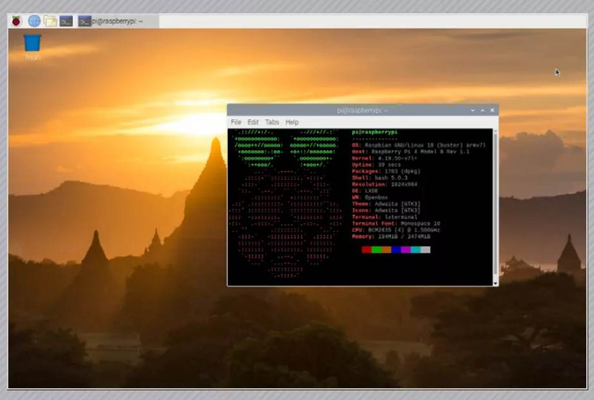
In short, the Pi 4 is the most powerful of the current Raspberry Pi models. However, the different memory versions have an increased cost. The 1GB version costs £34, 2GB is £44, and the 4GB version will set you back £54. Remember to also factor in one or two micro-HDMI cables with your order.

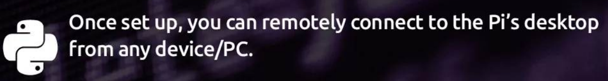
RASPBIAN BUSTER

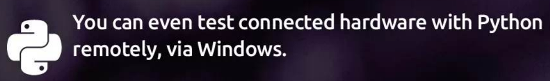
In addition to releasing the Pi 4, the Raspberry Pi team also compiled a new version of the Raspbian operating system, codenamed Buster.

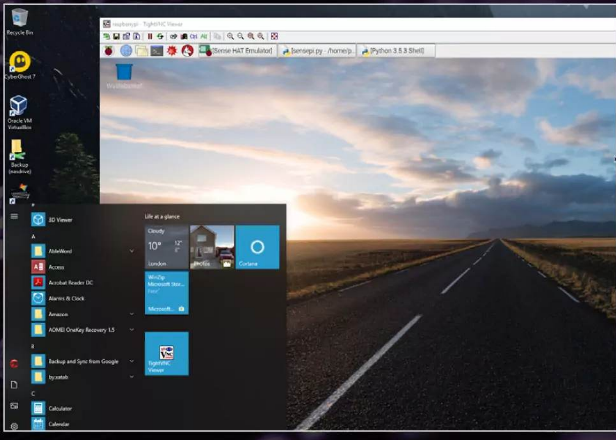
In conjunction with the new hardware the Pi 4 boasts, Buster does offer a few updates. Although on the whole it's very similar in appearance and operation to the previous version of Raspbian. The updates are mainly in-line with the 4K's display and playback, giving the Pi 4 a new set of graphical drivers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

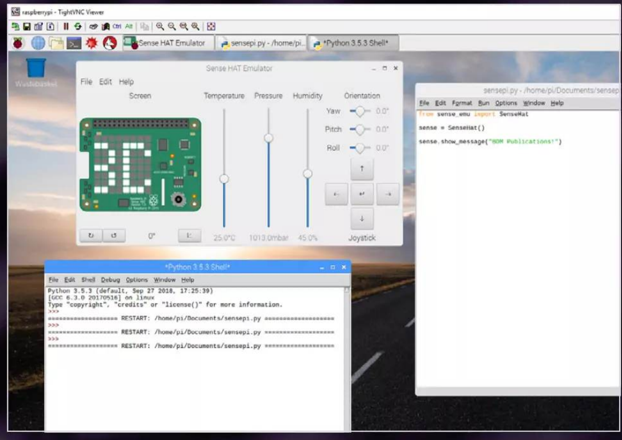
In short, what you see in this book will work with the Raspberry Pi 4 and Raspbian Buster!




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.





Using Virtual Machines

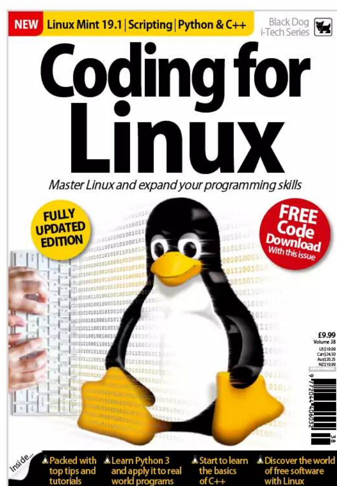
A Virtual Machine allows you to run an entire operating system from within an app on your desktop. This way, you're able to host multiple systems in a secure, safe and isolated environment. In short, it's an ideal way to code.

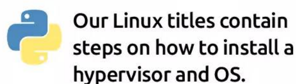
Sounds good, but what exactly is a Virtual Machine (VM) and how does it work?

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.

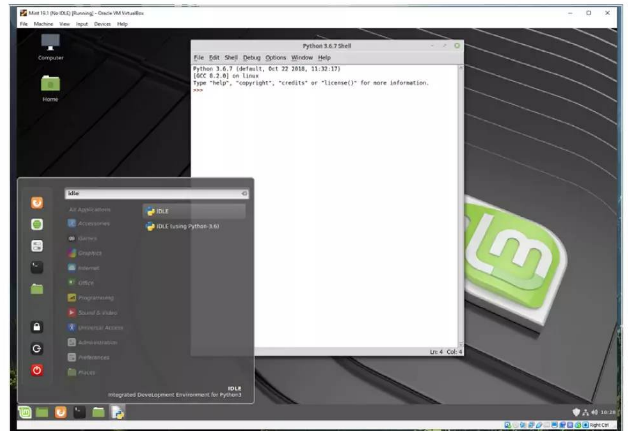
The Hypervisor will take resources from the host system - your physical computer, to create the virtual computer. This means that part of your physical computer's: memory, CPU, hard drive space and other shared resources, will be set aside for use in the virtual machine and therefore won't be available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

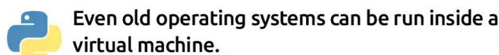
There are plenty of hypervisors available to try for free, with an equal amount commercially available that are significantly more powerful and offer better features. However, for most users, both beginner and professional, VirtualBox does a good enough job.

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).

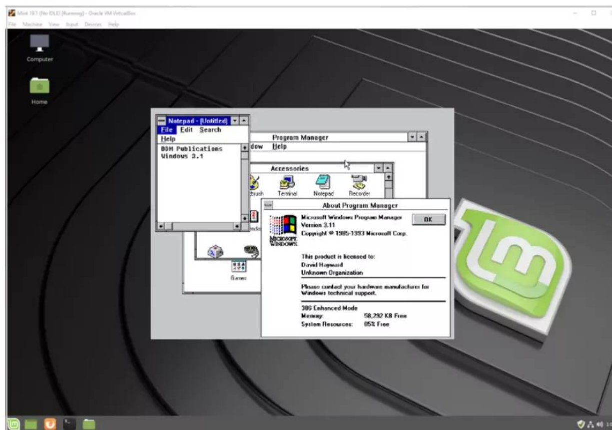
With this in mind then, you're able to have an installation of Linux Mint, or the latest version of Ubuntu, running in an app on your Windows 10 PC. This is the beauty of using a virtual machine. Conversely, if your physical computer has Linux as its installed operating system, then with a hypervisor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

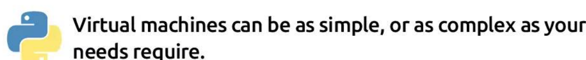
Using virtual machines removes the need to dual-boot. Dual-booting is having two, or more, physical operating systems installed on the same, or multiple, hard drives on a single computer. As the computer powers up, you're given the option to choose which OS you want to boot into. While this sounds like 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.

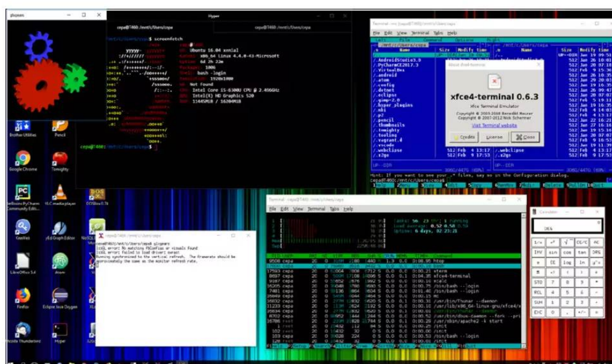


Even old operating systems can be run inside a virtual machine.





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

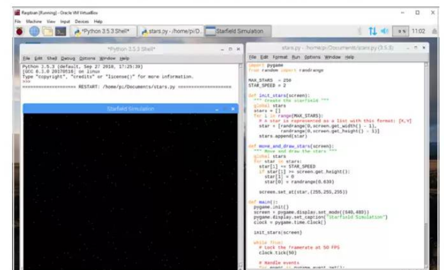


ADVANTAGES FOR CODERS

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 effect 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.

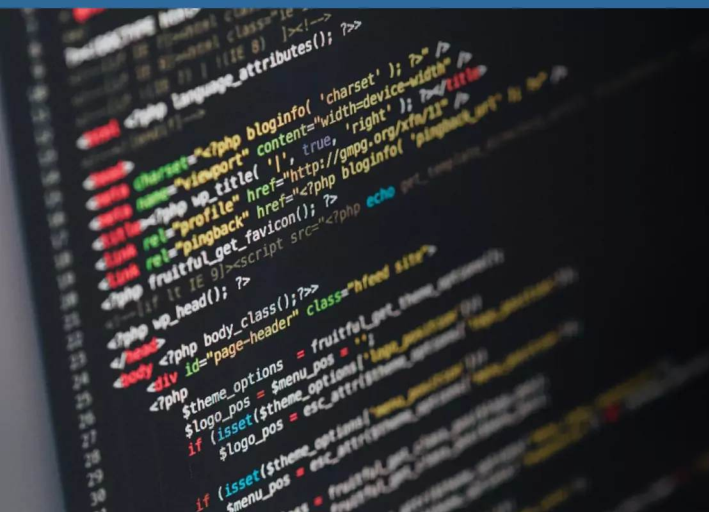
GOING VIRTUAL

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.

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.

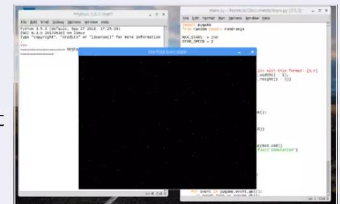
The trick is to slow down and, to begin with, not look too deeply into coding. Like all good projects, you need a solid foundation on which to build your skill and to have all the necessary tools available to hand to enable you to complete the basic steps. This is where creating a coding platform comes 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.

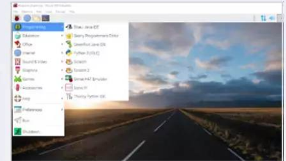
It's incredibly cheap, costing around £35 and allows you to utilise different hardware, in the form of robotics and electronics projects, as well as offering a complete desktop experience. Although not the most powerful computing device in the world, the Raspberry Pi has a lot going for it, especially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

Hello, World

Getting started with Python may seem a little daunting at first, but, thankfully, the language has been designed with simplicity in mind. Like most things, you need to start slow, master the basics, learn how to get a result, and how to get what you want from the code.

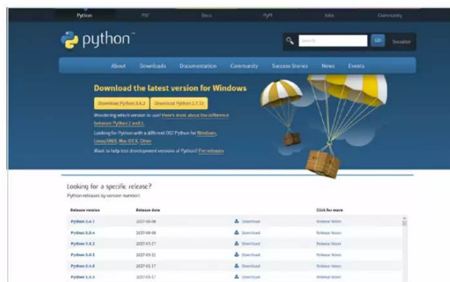
This section covers numbers and expressions, user input, conditions and loops and the types of errors you will undoubtedly come across in your time with Python: the core foundations of good coding and Python code.

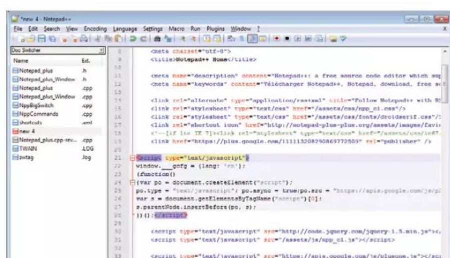
Equipment You Will Need

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

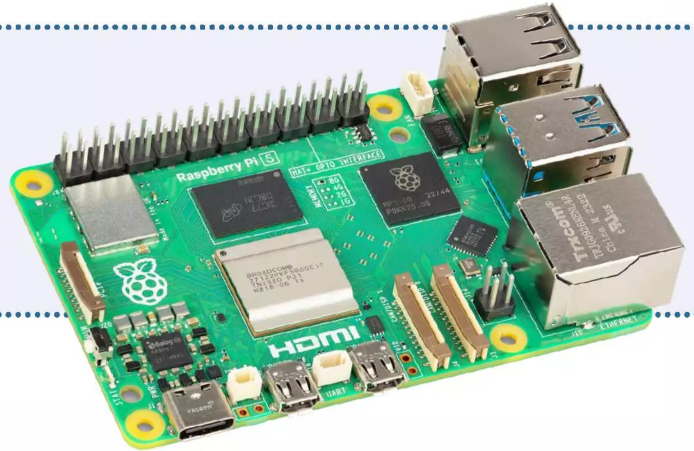
Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand you're learning something entirely new, and you will get there.

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 5 Model is the latest version, incorporating a more powerful CPU, a choice of 4GB or 8GB memory versions and Wi-Fi and Bluetooth support. You can pick up a Pi 5 from around £59, increasing up to £79 for the 8GB memory version, or as part of a kit depending on the `Pi model you're interested in.

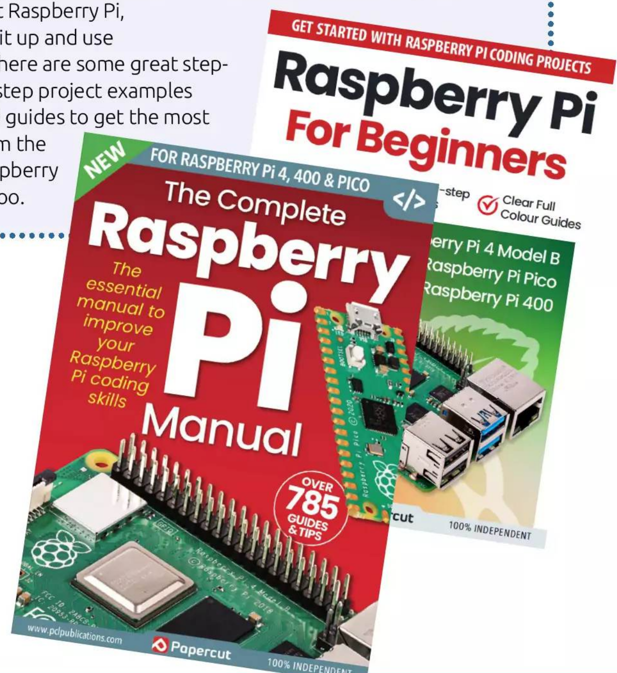


FUZE PROJECT

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

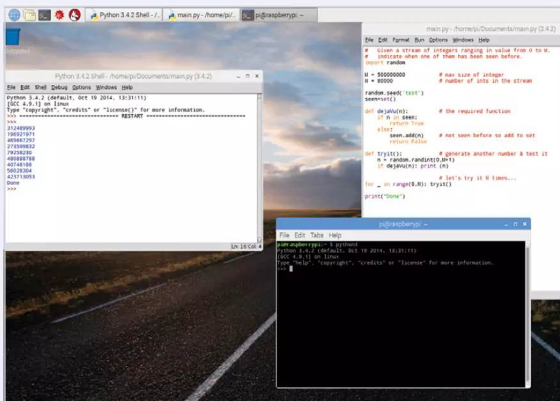
BOOKS

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



RASPBIAN

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



Getting to Know Python

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

WHAT IS PROGRAMMING?

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

PYTHON

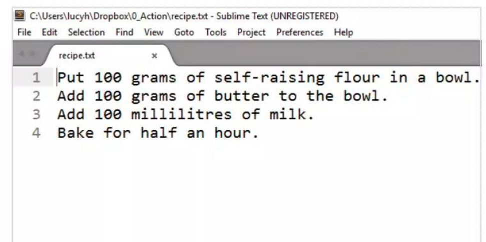
A programming language is a list of instructions that a computer follows. These instructions can be as simple as displaying your name or playing a music file, or as complex as building a whole virtual world. Python is a programming language conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde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

Guido van Rossum, the father of Python.

PROGRAMMING RECIPES

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

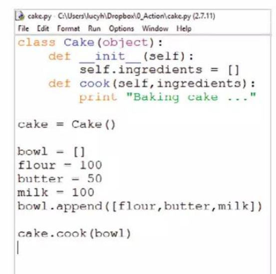
Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.



CODE

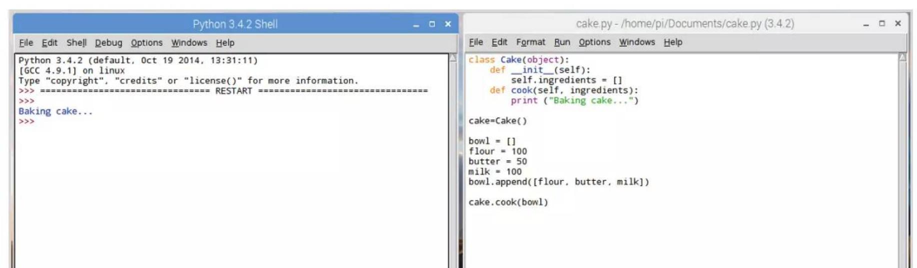
Just like a recipe, a program consists of instructions that you follow in order. A program that describes a cake might run like this:

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



PROGRAM COMMANDS

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

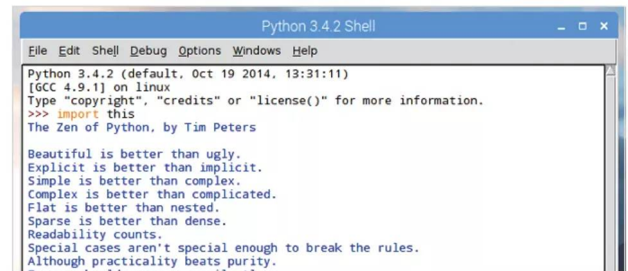


HIGH-LEVEL LANGUAGES

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

ZEN OF PYTHON

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



PYTHON 3 VS PYTHON 2

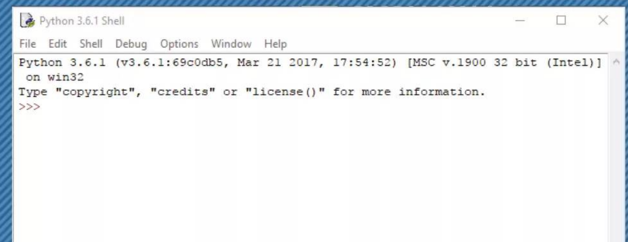
In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.

WORLD OF PYTHON

Python 3.7 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. 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.

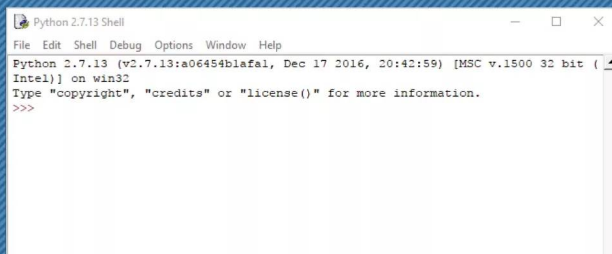
PYTHON 3.X

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



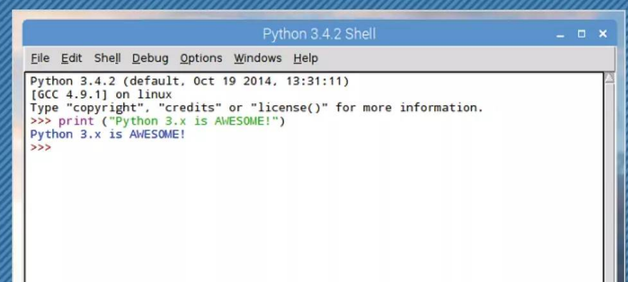
PYTHON 2.X

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



3.X WINS

Python 3's growing popularity has meant that it's now prudent to start learning to develop with the new features and begin to phase out the previous version. Many development companies, such as SpaceX and NASA use Python 3 for snippets of important code.



How to Set Up Python in Windows

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

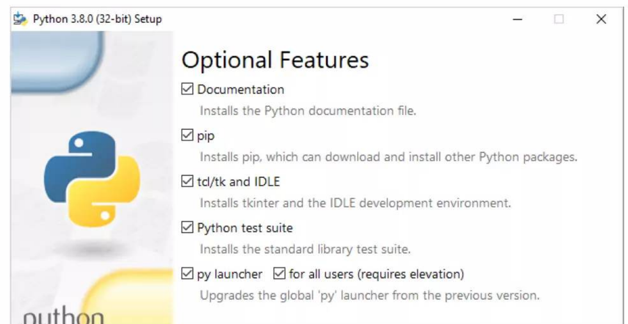
INSTALLING PYTHON 3.X

Microsoft Windows doesn't come with Python preinstalled as standard, so it will be necessary to install it yourself manually. Thankfully, it's an easy process to follow.

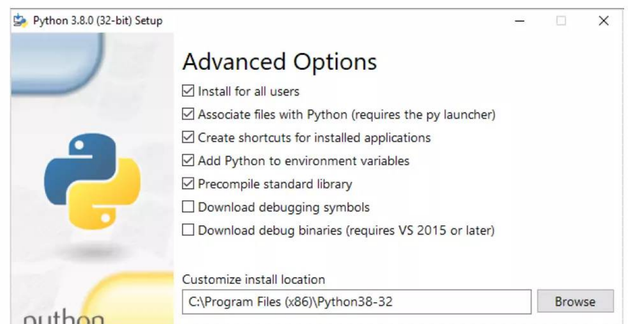
STEP 1 Start by opening your web browser to www.python.org/downloads/. Look for the button detailing the Download link for Python 3.x. Python is regularly updated, changing the last digit for each bug fix and update. Therefore, don't worry if you see Python 3.8, or more, as long as it's Python 3, the 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 have two choices: Install Now and Customise Installation. We recommend opting for the Customise Installation link.

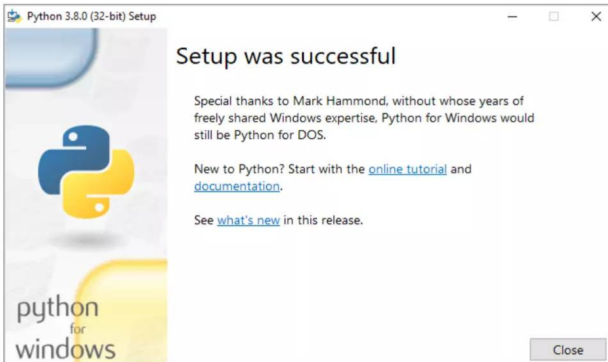
STEP 3 Choosing the Customise option allows you to specify certain parameters, and whilst you may stay with the defaults, it's a good habit to adopt as, sometimes (not with Python, thankfully), installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked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



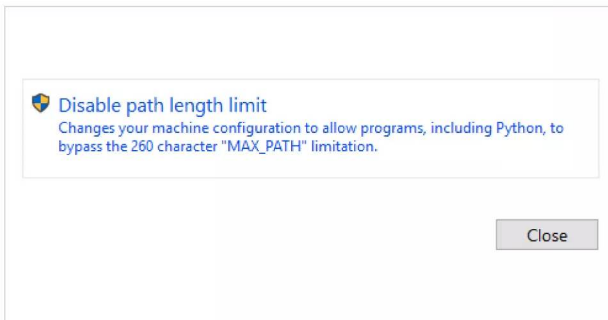
STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library and Install for All Users options are ticked. These make using Python later much easier. Click Install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



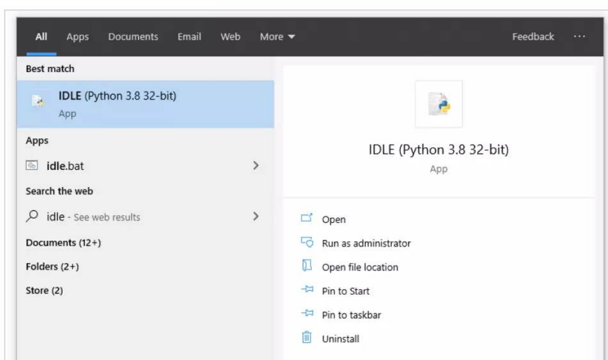
STEP 5 You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete, the final Python wizard page will allow you to view the latest release notes and follow some online tutorials.



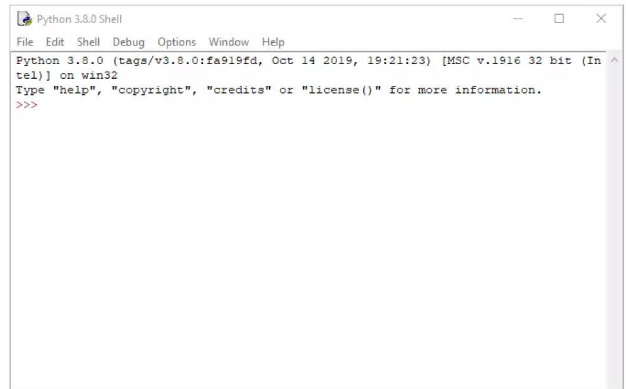
STEP 6 Before you close the install wizard window however, it's best to click on the link next to the shield detailed Disable Path Length Limit. This will allow Python to bypass the Windows 260 character limitation, enabling you to execute Python programs stored in deep folders arrangements. Click Yes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



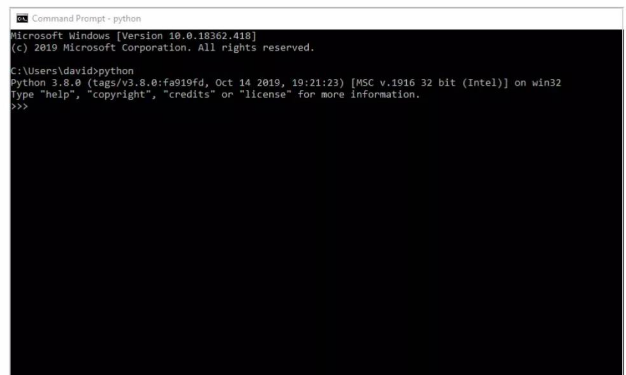
STEP 7 Windows 10 users can now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.x (32-bit) will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.



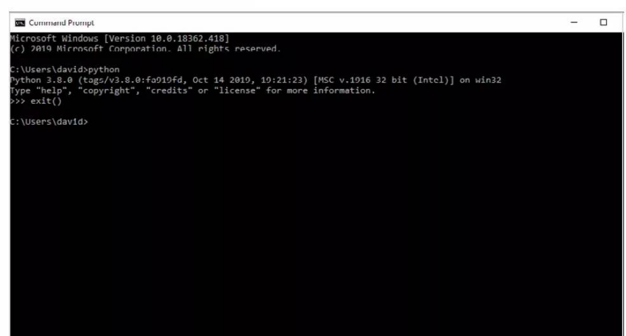
STEP 8 Clicking on the IDLE (Python 3.x 32-bit) link will launch the Python Shell, where you can begin your Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code works inside your Python 3 interface.



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



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



How to Set Up Python 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.

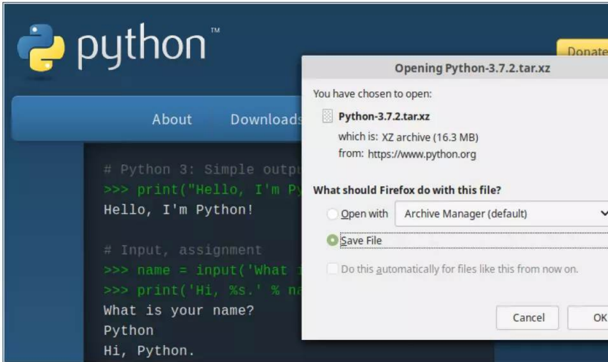
```
File  Edit  View  Search  Terminal  Help
Need to get 1,409 kB of archives.
After this operation, 23.6 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libasound2 amd6
4 1.1.3-5ubuntu0.2 [359 kB]
Get:2 http://archive.ubuntu.com/ubuntu 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) ...
```

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:

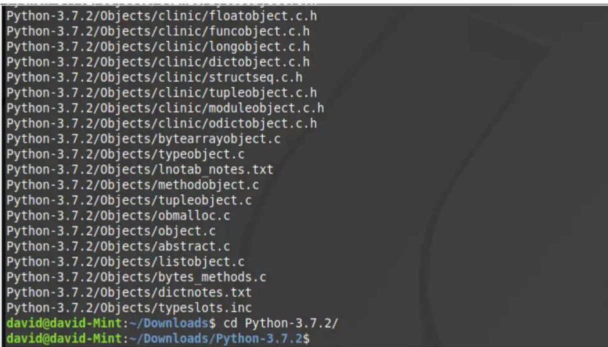
```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```

```
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.



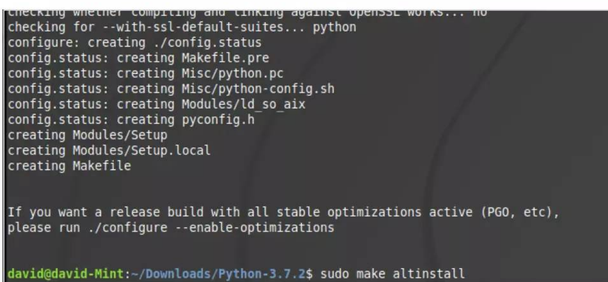
STEP 7 In the Terminal, go to the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.Y.Y.tar.xz` (replace the Y's with the version numbers you've downloaded). Now enter the newly unzipped folder with: `cd Python-3.Y.Y/`.



STEP 8 Within the Python folder, enter:

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

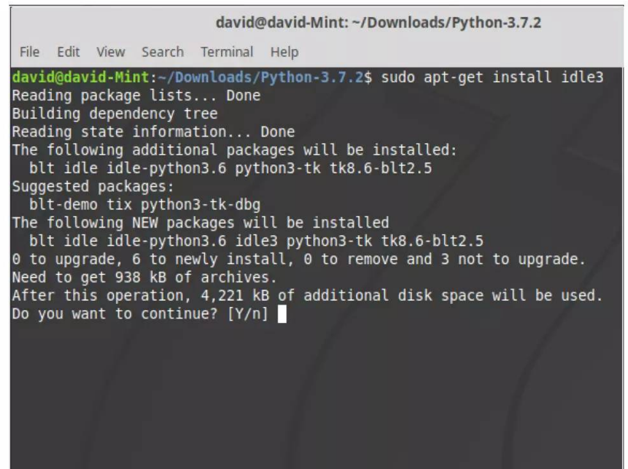
This could take a while, depending on the speed of your computer. Once finished, enter: `python3.7 --version` to check the latest installed version. You now have Python 3.7 installed, alongside older Python 3.x.x and Python 2.



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.

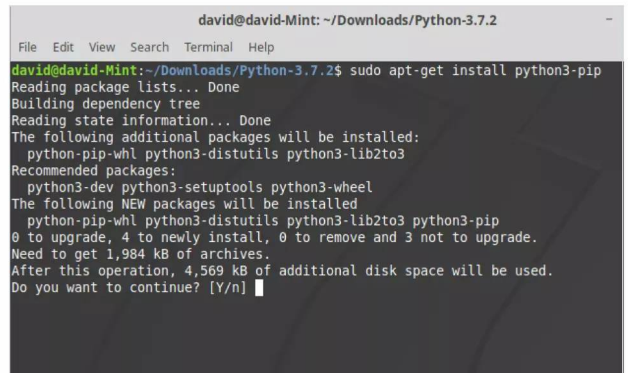


STEP 10 You'll also need PIP (Pip Installs Packages), which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

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.

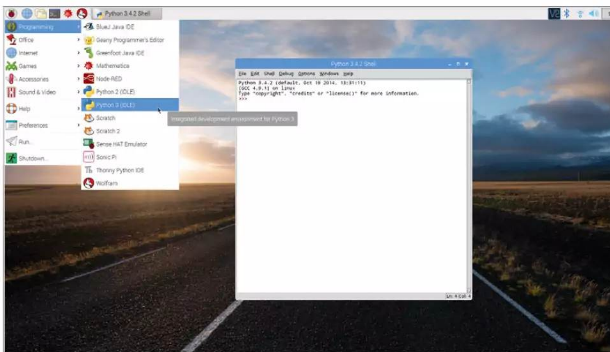
Starting Python for the First Time

The Raspberry Pi offers one of the best all-round solutions on which to learn and code, in particular, Python. Raspbian, the Pi's recommended OS, come pre-installed with the latest stable version of Python 3, which makes it a superb coding platform.

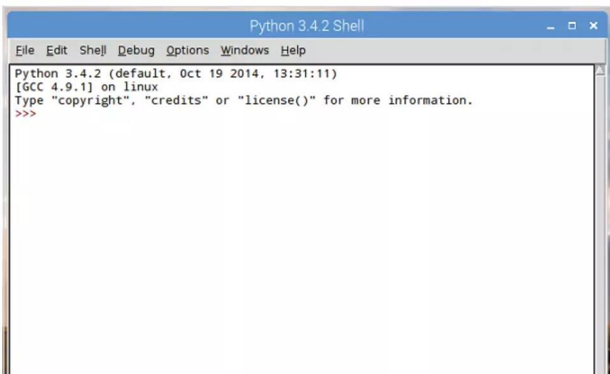
STARTING PYTHON

Everything you need to begin programming with Python is available from the Raspberry Pi desktop. However, if you want, drop into the Terminal and update the system with: sudo apt-get update.

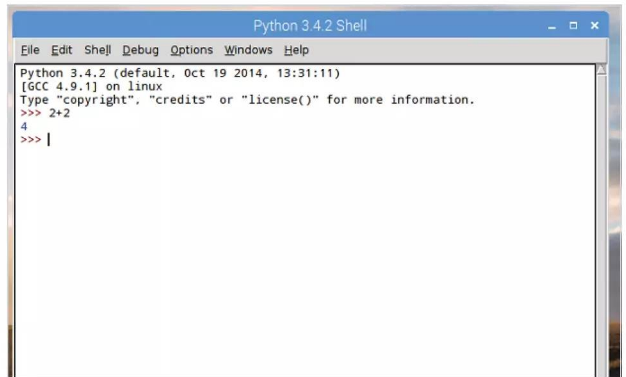
STEP 1 With the Raspbian desktop loaded, click on the Menu button followed by Programming > Python 3 (IDLE). This opens the Python 3 Shell. Windows and Mac users can find the Python 3 IDLE Shell from within the Windows Start button menu and via Finder.



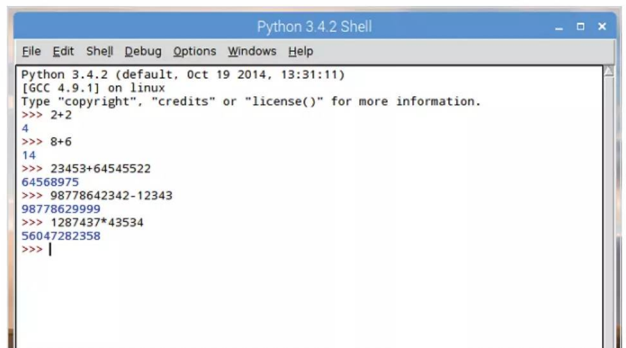
STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, where you're able to try out some simple code and processes.



STEP 3 For example, in the Shell enter: 2+2 After pressing Enter, the next line displays the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.



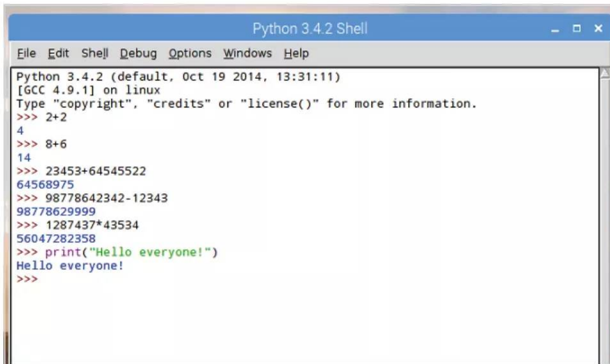
STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers can easily be added, subtracted, multiplied and so on.



STEP 5 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.



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

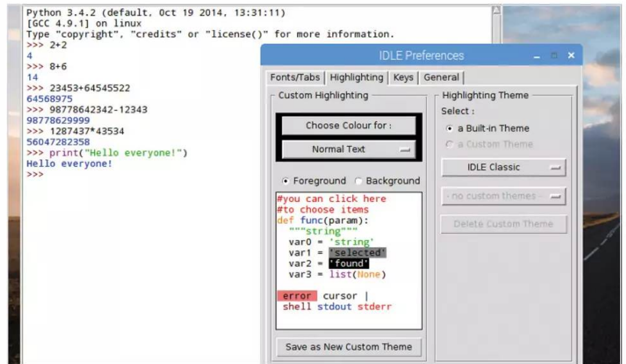
STEP 7 You may have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

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

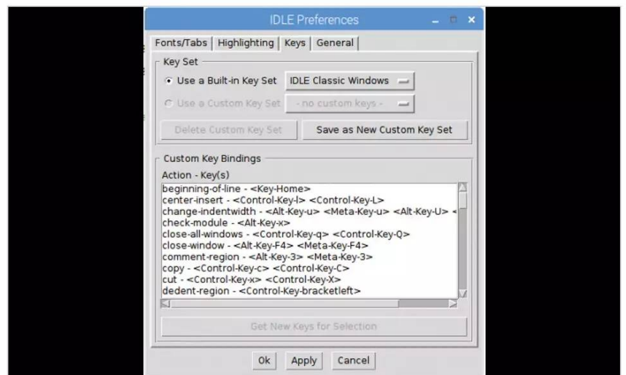
IDLE Colour Coding

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

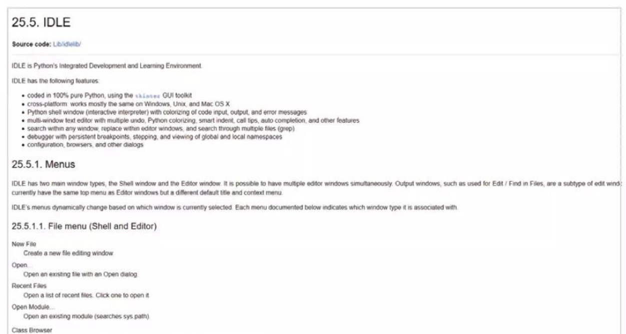
STEP 8 The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via Options > Configure IDLE and clicking on the Highlighting tab. However, we don't recommend that, as you won't be seeing the same as our screenshots.



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



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



Your First Code

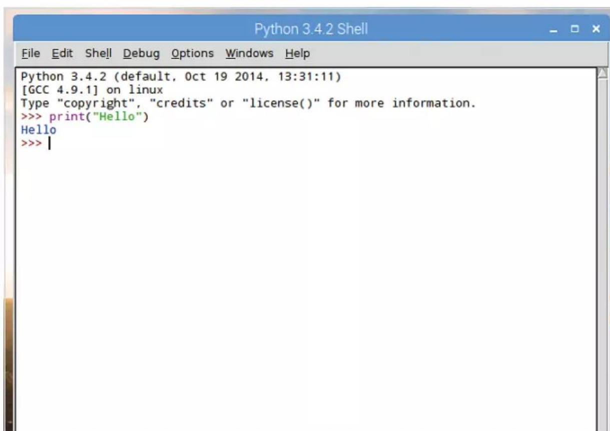
Essentially, you've already written your first piece of code with the 'print("Hello everyone!")' function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

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

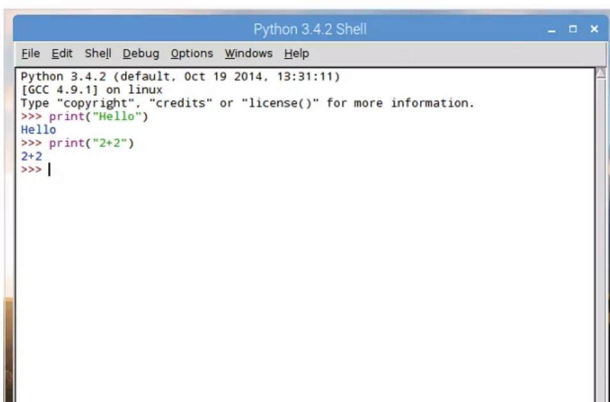
STEP 1 If you've closed Python 3 IDLE, reopen it in whichever operating system version you prefer. In the Shell, enter the familiar following:

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



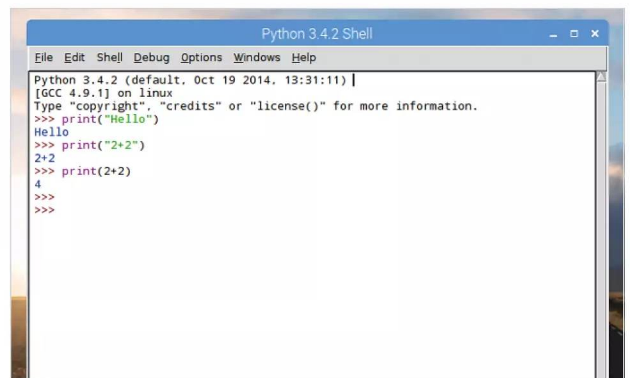
STEP 2 Just as predicted, the word Hello appears in the Shell as blue text, indicating output from a string. It's fairly straightforward and doesn't require too much explanation. Now try:

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



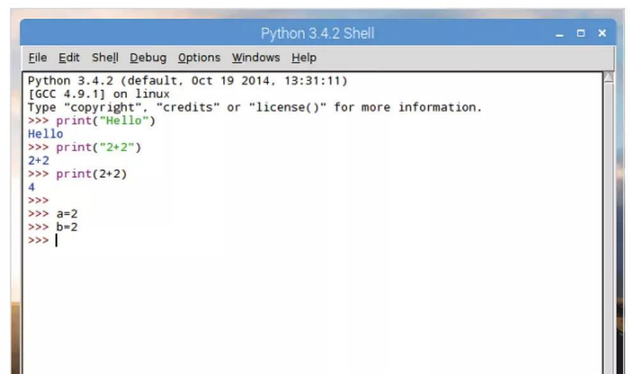
STEP 3 You can see that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell; to print the total of 2+2 you need to remove the quotes:

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



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

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



STEP 5

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 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 both a and b individually, as you've asked them to be printed separately. If you want to add them up, you can use the following:

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

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

```
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 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 and the Print function. For example, you could assign variables for someone's name:

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

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

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

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> |
```

STEP 10

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

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

Alternatively, you can add the space yourself:

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> print(name, surname)
David Hayward
>>> print(name+" "+surname)
David Hayward
>>> |
```

Saving and Executing Your Code

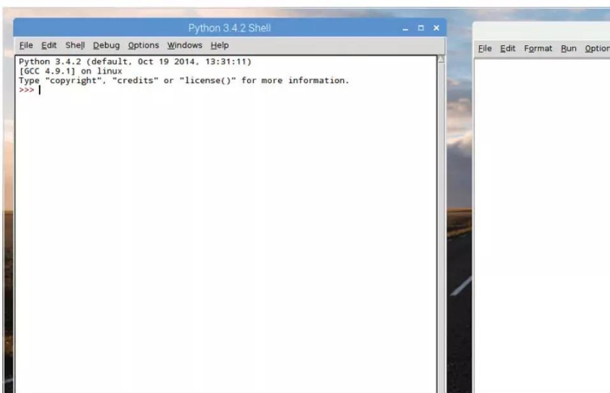
While working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

EDITING CODE

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

STEP 1

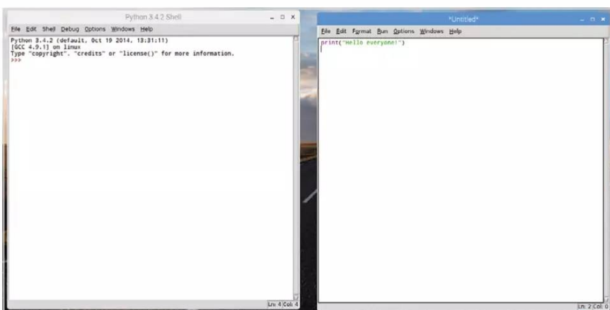
First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.



STEP 2

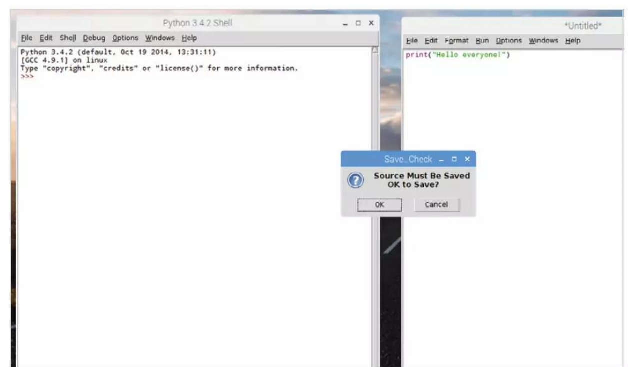
The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

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



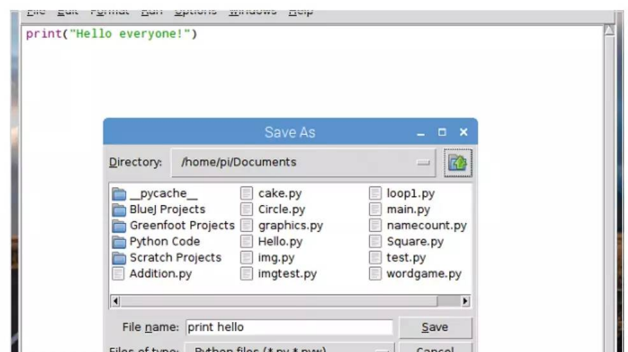
STEP 3

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



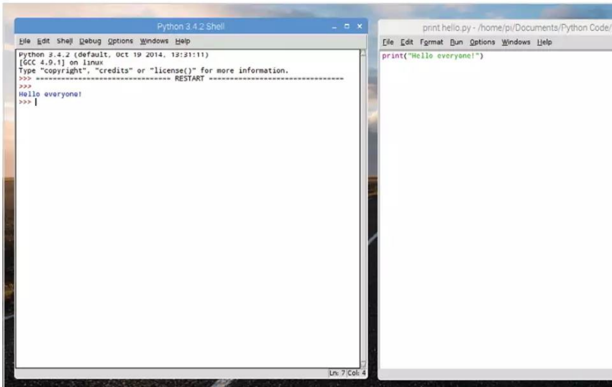
STEP 4

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



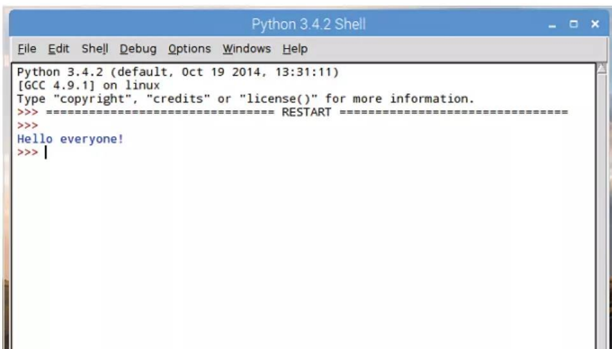
STEP 5

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



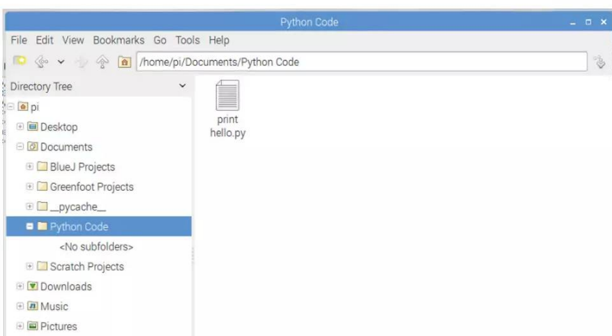
STEP 6

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



STEP 7

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

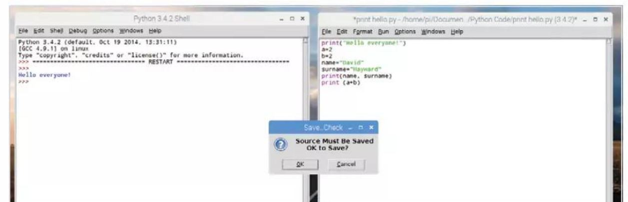


STEP 8

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

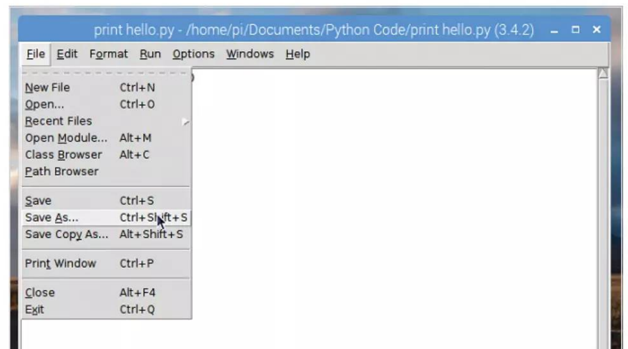
```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

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



STEP 9

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

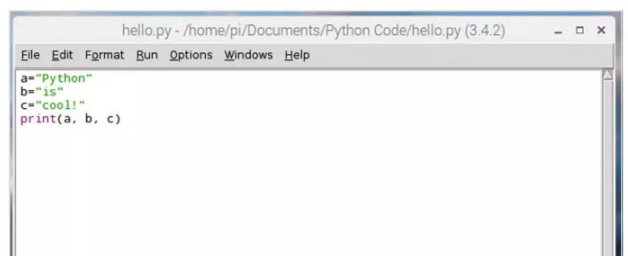


STEP 10

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

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

You will use this code in the next tutorial.



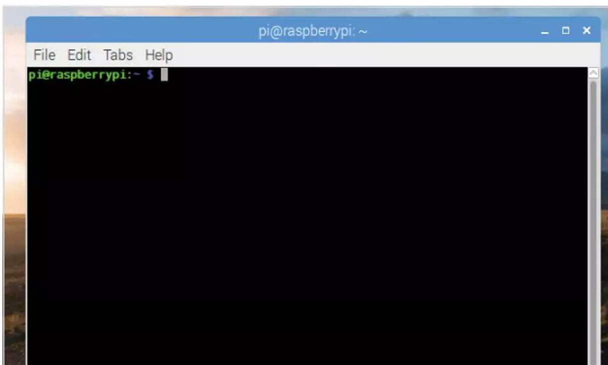
Executing Code from the Command Line

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

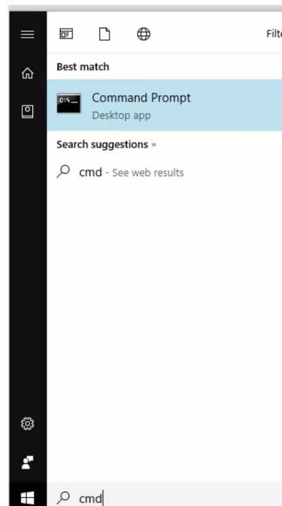
COMMAND THE CODE

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

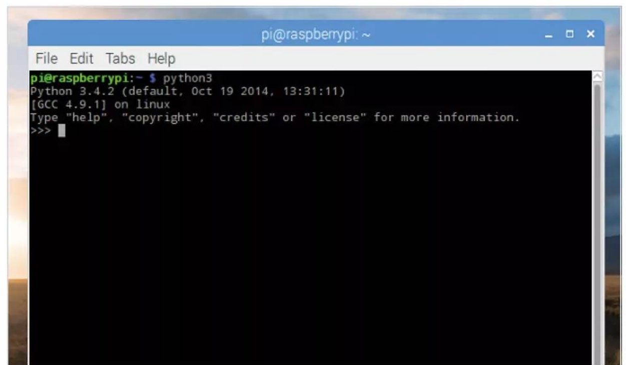
STEP 1 Python, in Linux, comes with two possible ways of executing code via the command line. One of the ways is with Python 2, whilst the other uses the Python 3 libraries and so on. First though, drop into the command line or Terminal on your operating system.



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



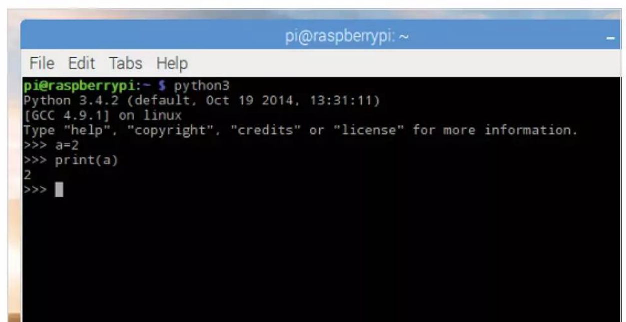
STEP 3 Now you're at the command line we can start Python. For Python 3 you need to enter the command **python3** and press Enter. This will put you into the command line version of the Shell, with the familiar three right-facing arrows as the cursor (>>>).



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

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

You can see that it works exactly the same.



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

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

```
python3 hello.py
```

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

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

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

```
python hello.py
```

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

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

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

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

```
python3 hello.py
```

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

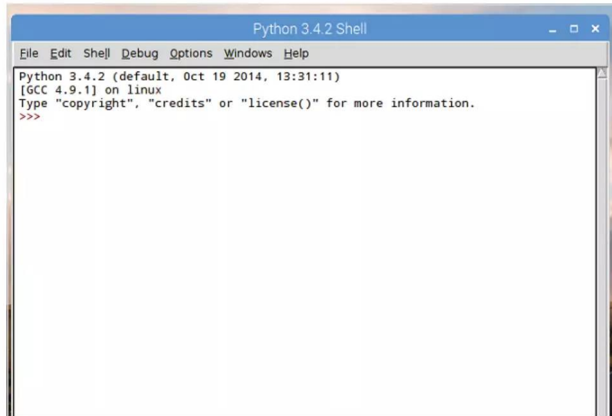
Numbers and Expressions

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

IT'S ALL MATHS, MAN

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

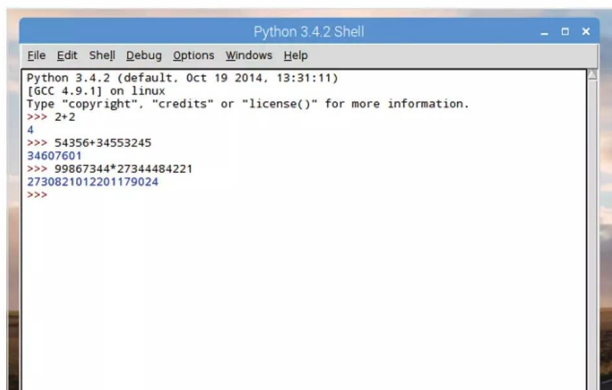
STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, you're going to use the Shell just to warm our maths muscle, which we believe is a small gland located at the back of the brain (or not).



STEP 2 In the Shell enter the following:

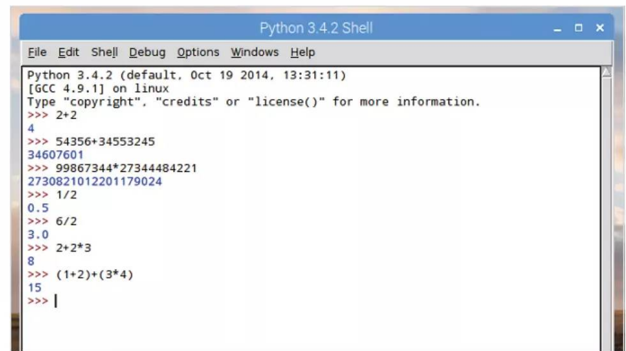
```
2+2
54356+34553245
99867344*27344484221
```

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



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

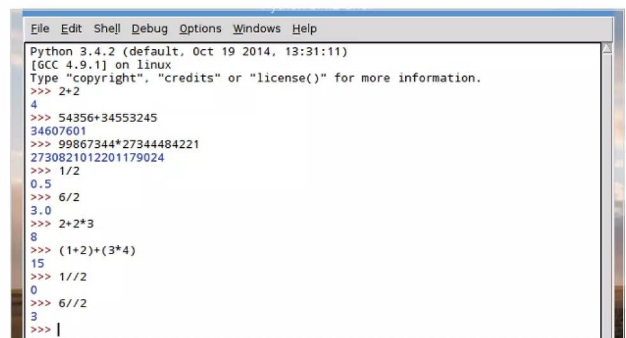
```
1/2
6/2
2+2*3
(1+2)+(3*4)
```



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

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

And so on.



STEP 5

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

```
10/3
```

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

```
10%3
```

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

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

STEP 6

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

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

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

```
>>> 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)
```

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

STEP 8

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

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

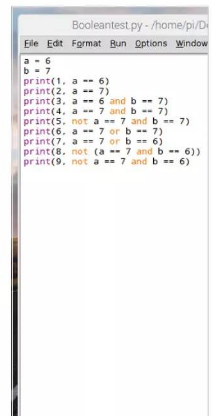
The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

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

STEP 9

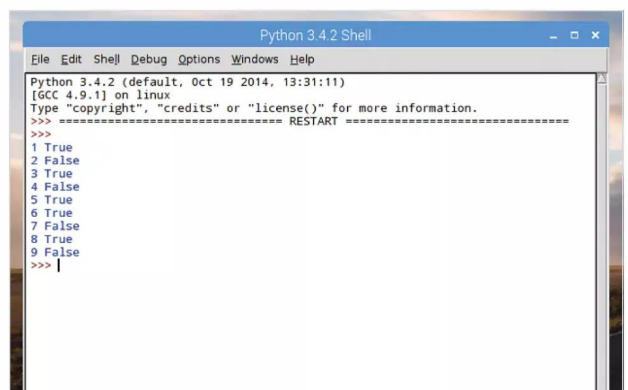
A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data and test to see if it's equal to, less than or greater than. Try this in a New File:

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



STEP 10

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



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 over time, it can become difficult to read.

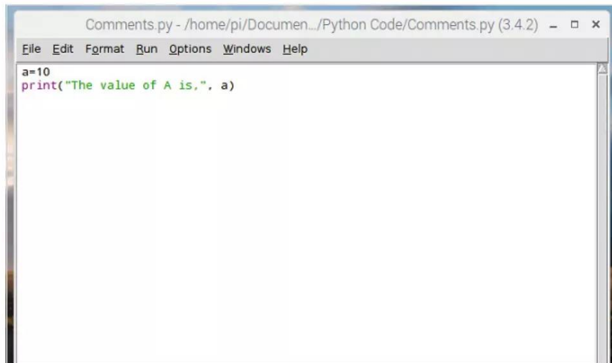
#COMMENTS!

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

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

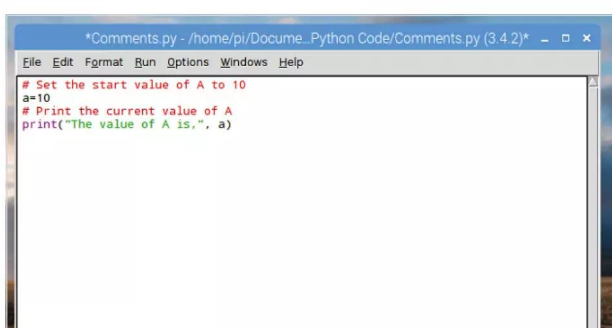
```
a=10
print("The value of A is,", a)
```

Save the file and execute the code.

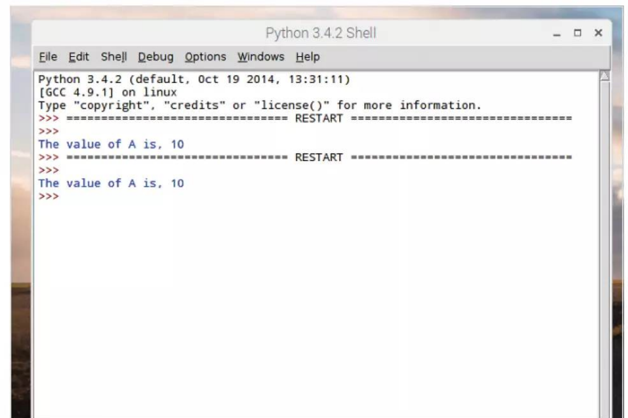


STEP 2 Running the code will return the line: The value of A is, 10 into the IDLE Shell window, which is what we expected. Now, add some of the types of comments you'd normally see within code:

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

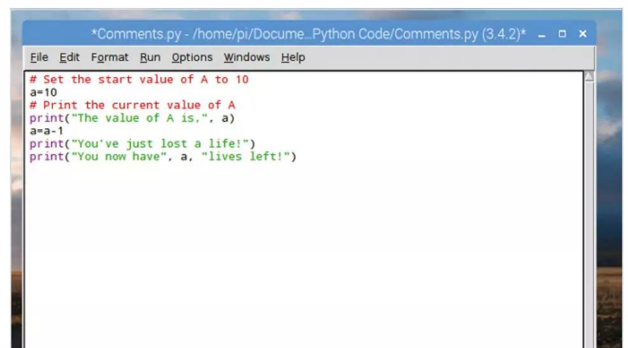


STEP 3 Resave the code and execute it. You can see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert to inform them, and others, of what's going on without the user being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



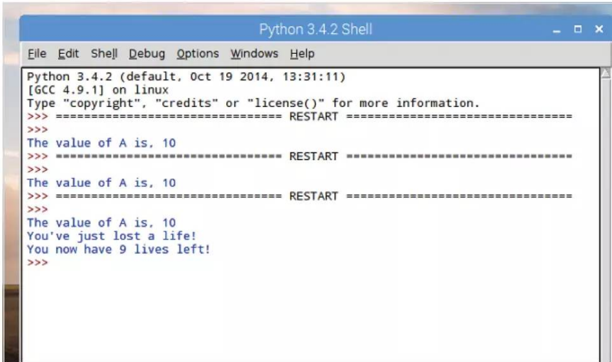
STEP 4 Let's assume that the variable A that we've created is the number of lives in a game. Every time the player dies, the value is decreased by 1. The programmer could insert a routine along the lines of:

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



STEP 5

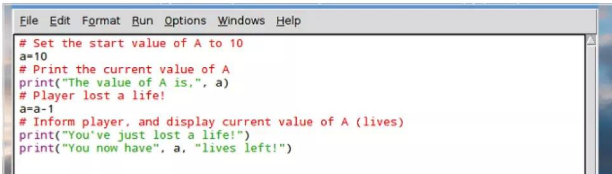
Whilst we know that the variable A is lives, and that the player has just lost one, a casual viewer or someone checking the code may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.



STEP 6

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

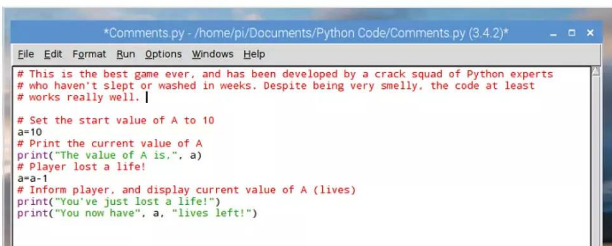
```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A
(lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```



STEP 7

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

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



STEP 8

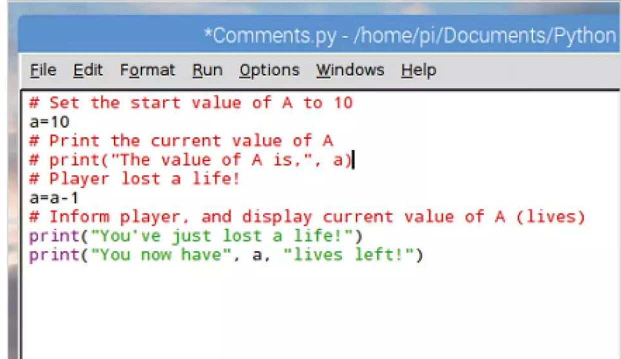
Inline comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

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

STEP 9

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

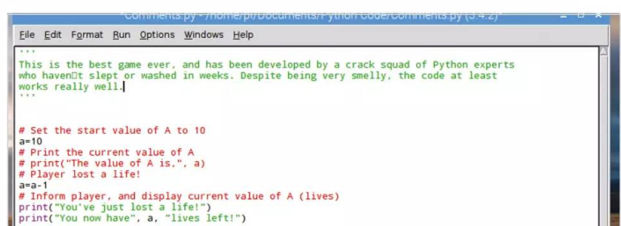
```
# print("The value of A is,", a)
```



STEP 10

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

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



Working with Variables

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

VARIOUS VARIABLES

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

STEP 1

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>>
```

STEP 2

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> |
```

STEP 3

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> |
```

STEP 4

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

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

Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

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

Which, as we know, are integers.

```
Python 3.4
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:1
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>>
```

STEP 5

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

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

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

STEP 6

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

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

or you can use:

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

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

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

STEP 7

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

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

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

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

STEP 8

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

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

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

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

STEP 9

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

```
shirt=19.99
```

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

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

>>>
```

STEP 10

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

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

User Input

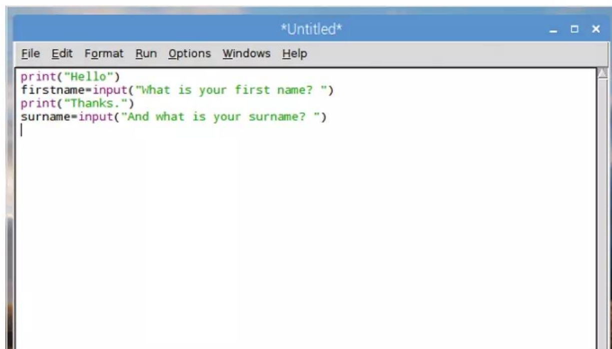
We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you would get information from the user then store and present it.

USER FRIENDLY

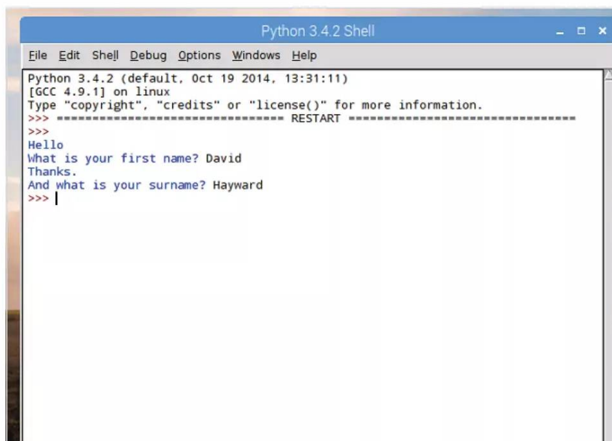
The type of input you want from the user will depend greatly on the type of program you're coding. For example, a game may ask for a character's name, whereas a database can ask for personal details.

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

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

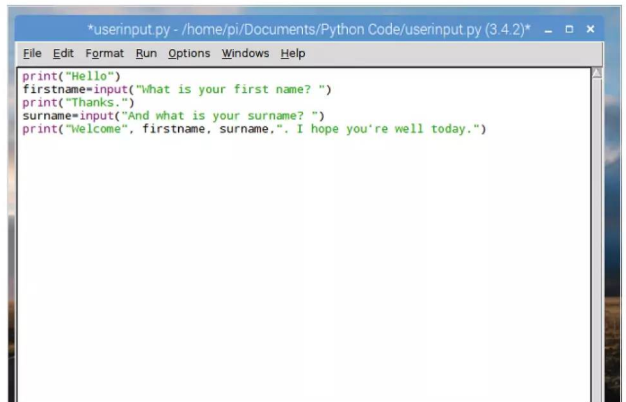


STEP 2 Save and execute the code, and as you already no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable firstname, followed by your surname; also stored in its own variable (surname).



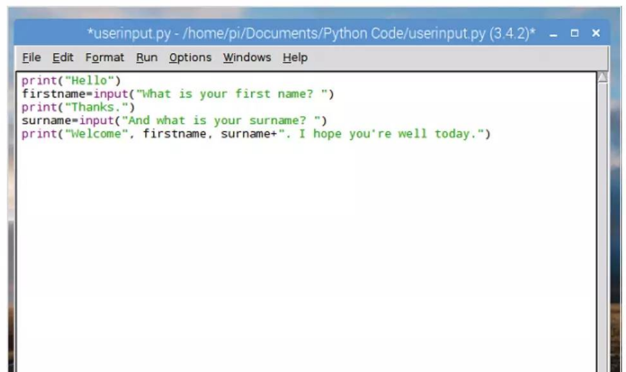
STEP 3 Now that we have the user's name stored in a couple of variables we can call them up whenever we want:

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



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

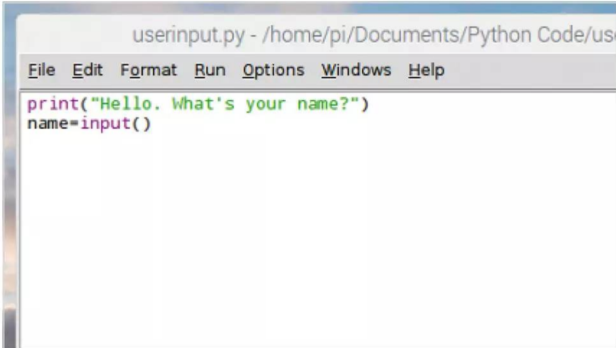
```
print("Welcome", firstname, surname+". I hope you're well today.")
```



STEP 5

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

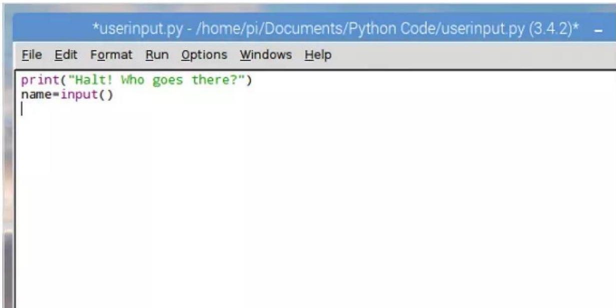
```
print("Hello. What's your name?")
name=input()
```



STEP 6

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

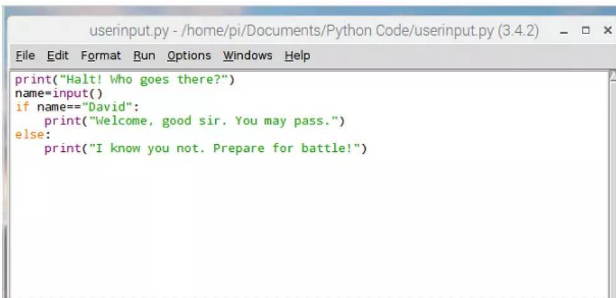
```
print("Halt! Who goes there?")
name=input()
```



STEP 7

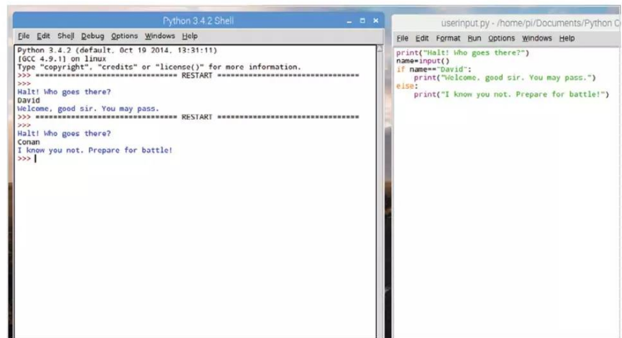
It's a good start to a text adventure game, perhaps? Now you can expand on it and use the raw input from the user to flesh out the game a little:

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



STEP 8

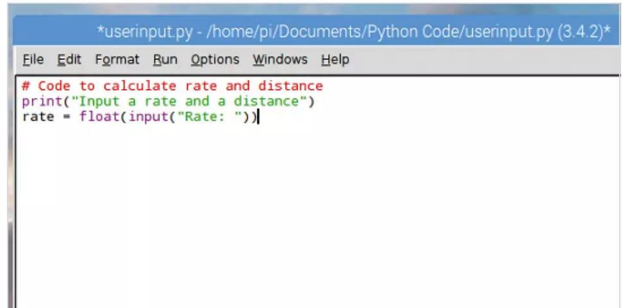
What you've created here is a condition, which we will cover soon. In short, we're using the input from the user and measuring it against a condition. So, if the user enters David as their name, the guard will allow them to pass unhindered. Else, if they enter a name other than David, the guard challenges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



STEP 9

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

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

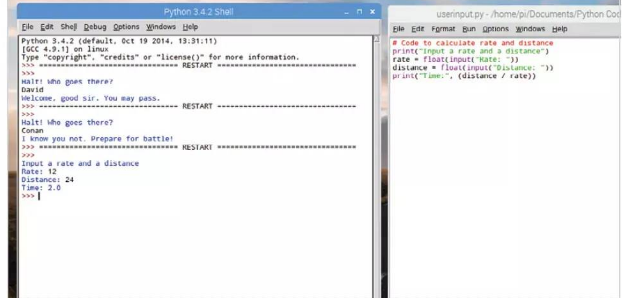


STEP 10

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

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

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



Creating Functions

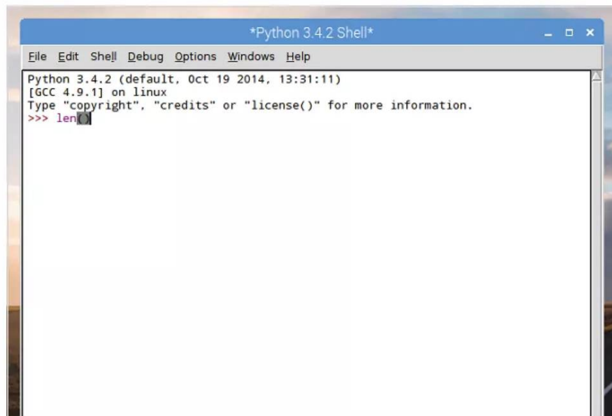
Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command but Python enables you to define your own functions.

FUNKY FUNCTIONS

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

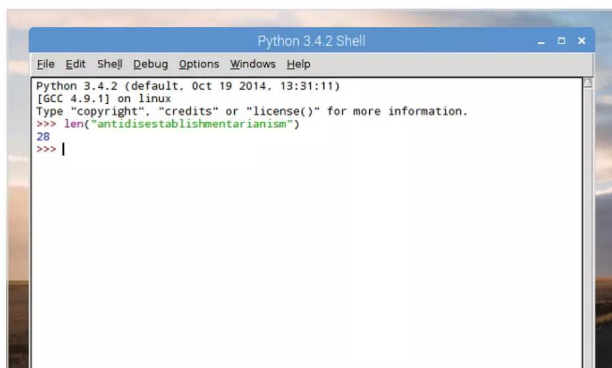
STEP 1

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



STEP 2

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

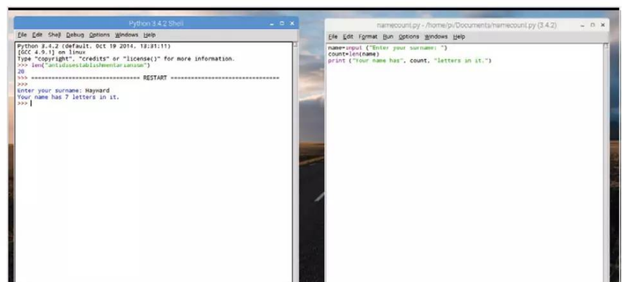


STEP 3

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

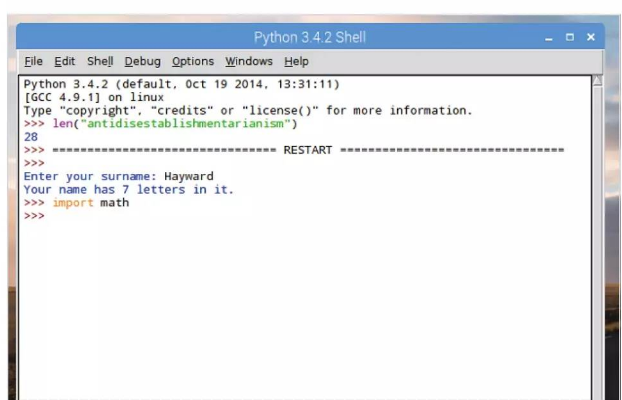
```
name=input ("Enter your surname: ")
count=len(name)
print ("Your surname has", count, "letters in it.")
```

Press F5 and save the code to execute it.



STEP 4

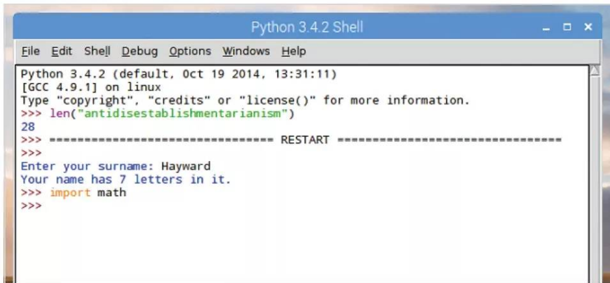
Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to www.docs.python.org/3/library/functions.html. These are the predefined functions, but since users have created many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



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

```
import math
```

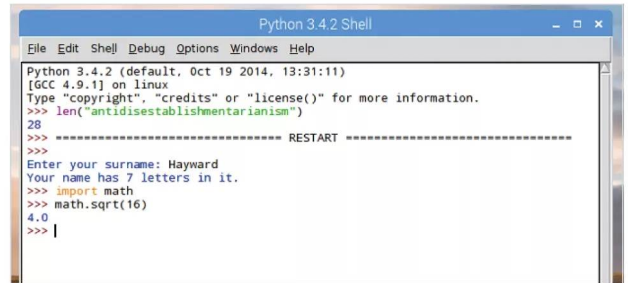
Once entered, you have access to all the Math module functions.



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

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

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



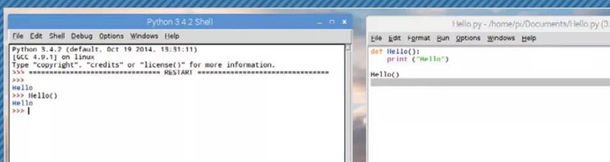
FORGING FUNCTIONS

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

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

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

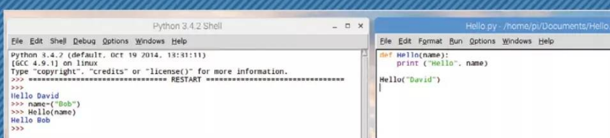
Press F5 to save and run the script. You can see Hello in the Shell, type in Hello() and it returns the new function.



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

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

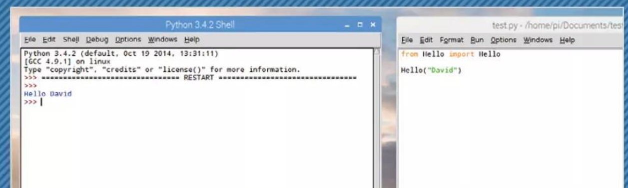
This will now accept the variable name, otherwise it prints Hello David. In the Shell, enter: name=("Bob"), then: Hello(name). Your function can now pass variables through it.



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

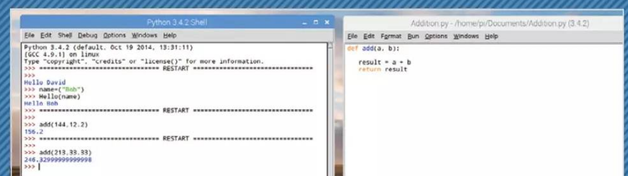
```
from Hello import Hello
Hello("David")
```

Press F5 to save and execute the code.



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

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



Conditions and Loops

Conditions and loops are what 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.

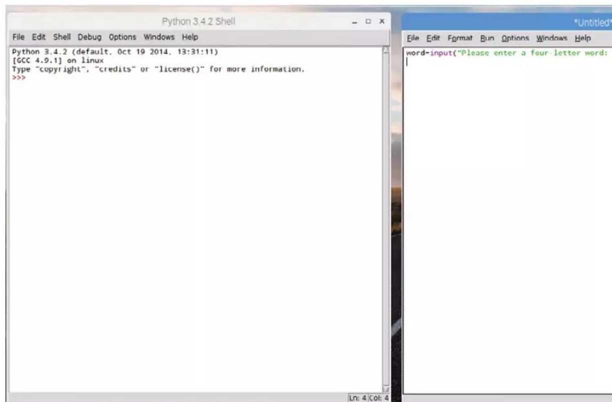
TRUE CONDITIONS

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:

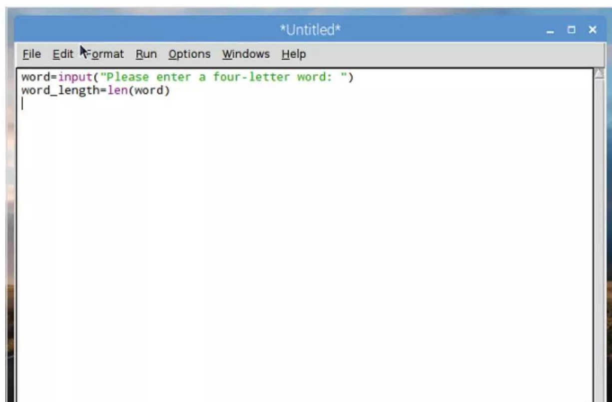
```
word=input("Please enter a four-letter word: ")
```



STEP 2

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

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

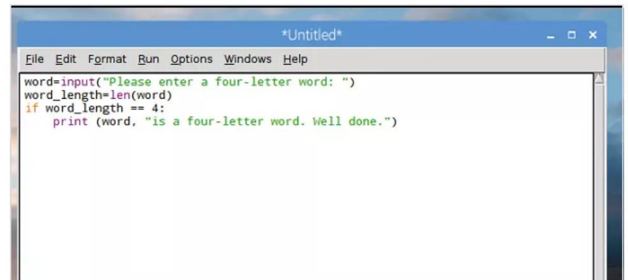


STEP 3

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

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

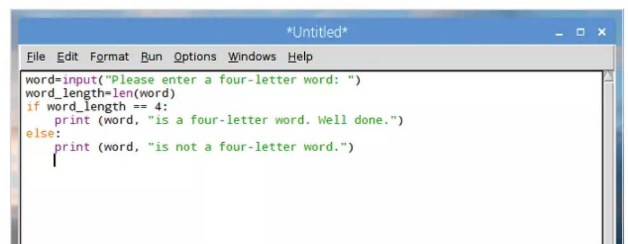
The double equal sign (==) means check if something is equal to something else.



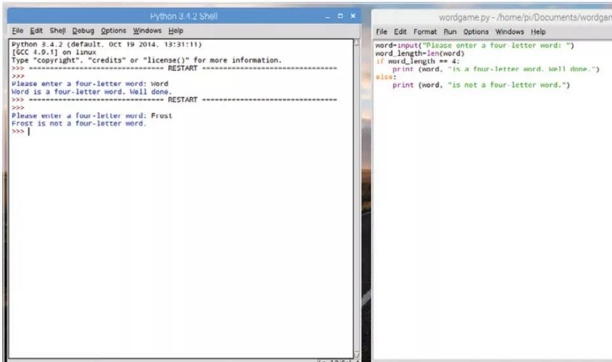
STEP 4

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

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

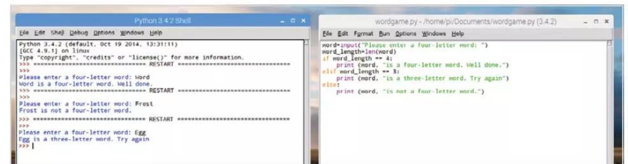


STEP 5 Press F5 and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that it's the word is four letters. Now press F5 again and rerun the program but this time enter a five-letter word. The Shell will display that it's not a four-letter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



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

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

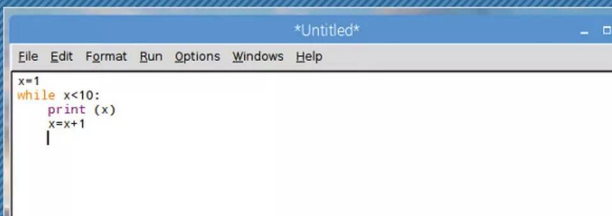


LOOPS

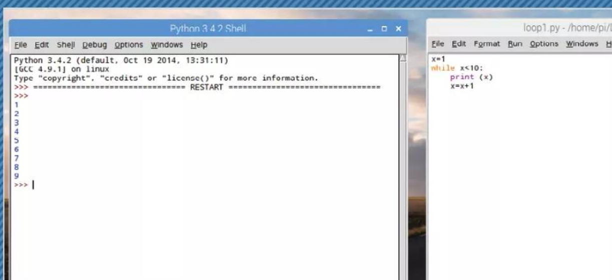
A loop looks quite similar to a condition but they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

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

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

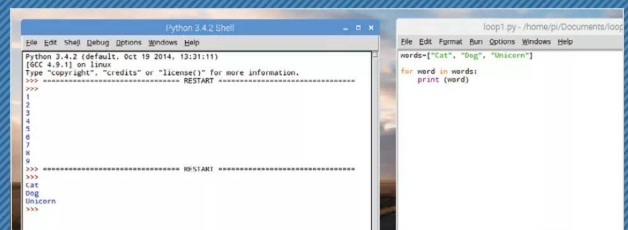


STEP 2 The difference between if and while is when while gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop it prints the current value of x, then adds one to that value. When x does eventually equal 10 it stops.



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

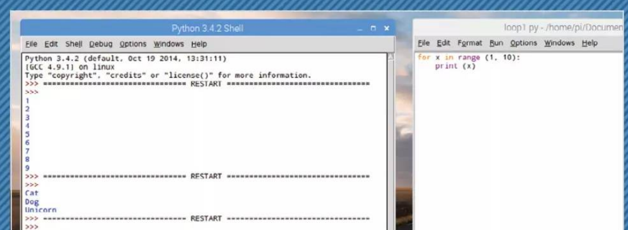
```
words=["Cat", "Dog", "Unicorn"]

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



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

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

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



Python Modules

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

MASTERING MODULES

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

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

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>>
```

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

```
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> 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
```

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

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

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

```
Python 3.6.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>>
```

STEP 6

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

```
import random
```

Brings in the code from the Random Number Generator module. You can then use this module to create something like:

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

```
*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 great or lesser range. For example:

```
import random

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

```
Python 3.6.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>>
========== RESTART: C:/Users/david/Documents/Python/Rnd Number.py ==========
```

```
Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py
File  Edit  Format  Run  Options  Window  Help
import random

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

STEP 8

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

```
import random
import math

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

print(math.pi)
```

```
Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
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:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
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:/Users/david/Documents/Python/Rnd Number.py (3.6.2)*
File  Edit  Format  Run  Options  Window  Help
import math as m

print(m.pi)
```

Python Errors

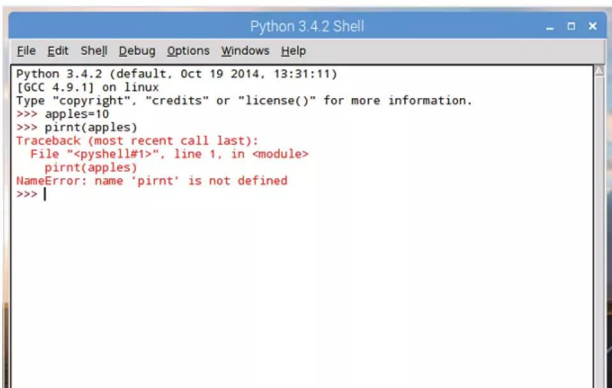
It goes without saying that you'll eventually come across an error in your code, where Python declares it's not able to continue due to something being missed out, wrong or simply unknown. Being able to identify these errors makes for a good programmer.

DEBUGGING

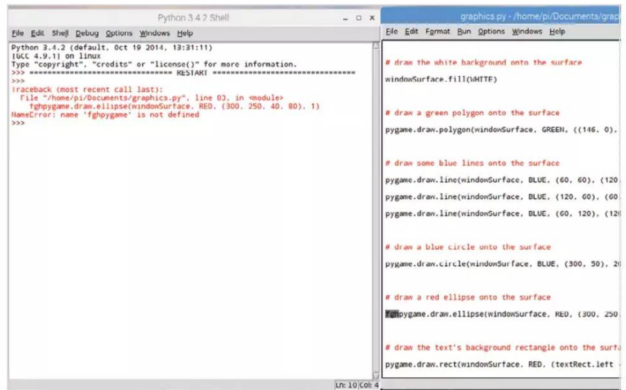
Errors in code are called bugs and are perfectly normal. They can often be easily rectified with a little patience. The important thing is to keep looking, experimenting and testing. Eventually your code will be bug free.

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

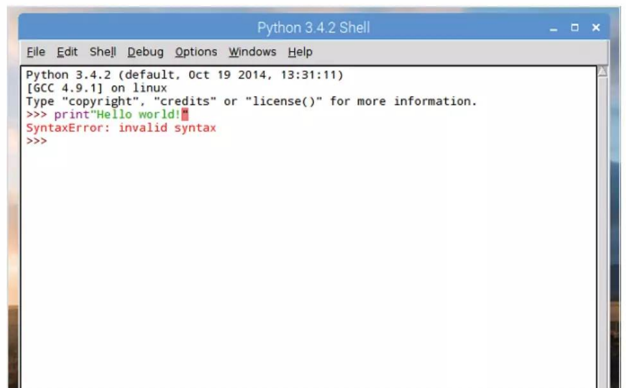
STEP 2 The most common of errors is the typo, as we've mentioned. The typos are often at the command level: mistyping the print command for example. However, they also occur when you have numerous variables, all of which have lengthy names. The best advice is to simply go through the code and check your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



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



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



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

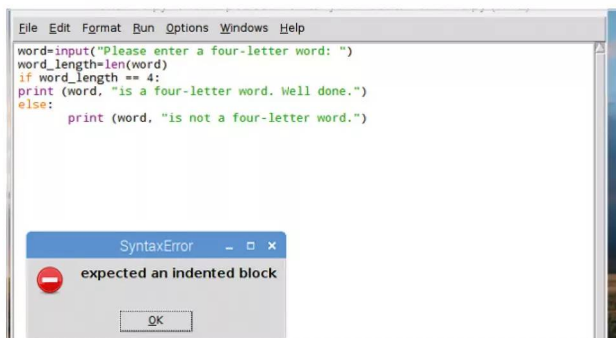
```
print(balanced_check(input()))
```

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

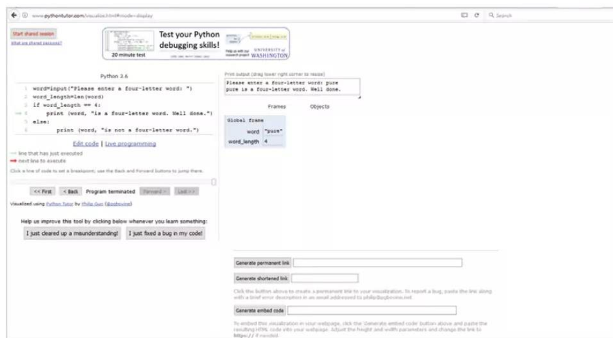
```
1  import sys
2
3  def balanced_check(data):
4      stack = []
5      characters = list(data)
6
7      for character in characters:
8          reference = {
9              '(': ')',
10             '{': '}',
11             '[': ']'
12         }
13         if character in reference.keys():
14             stack.append(character)
15
16         elif character in reference.values() and len(stack) > 0:
17             char = stack.pop()
18             if reference.get(char) != character:
19                 return "NO"
20         else:
```

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

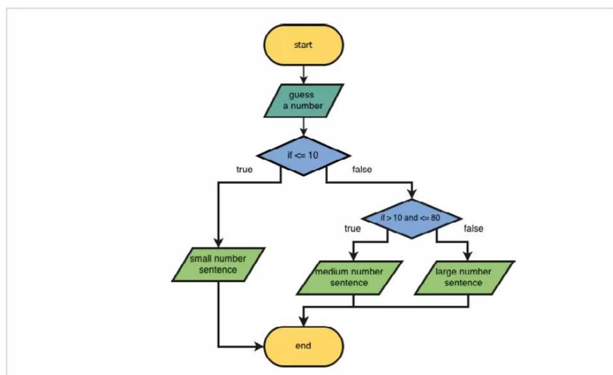
STEP 7 Indents are a nasty part of Python programming that a lot of beginners fall foul of. Recall the If loop from the Conditions and Loops section, where the colon means everything indented following the statement is to be executed as long as it's true? Missing the indent, or having too much of indent,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



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



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



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

Combining What You Know So Far

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

PLAYING WITH PI

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

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

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

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

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

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

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

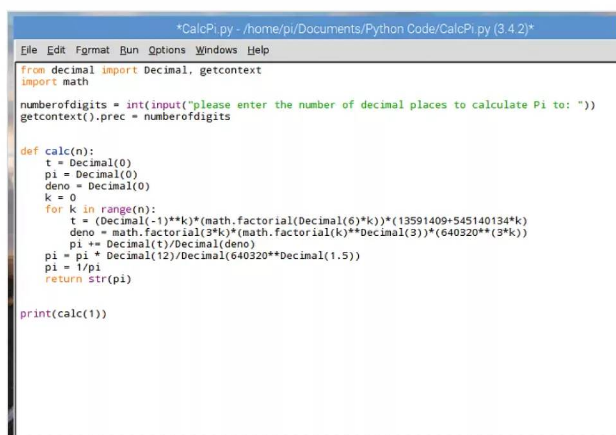
STEP 4 The previous step defines the rules that make up the algorithm and creates the string that will eventually display the value of Pi, according the Chudnovsky brothers' algorithm. You have no doubt already surmised that it would be handy to actually output the value of Pi to the screen. To rectify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

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

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

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

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



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

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

```
pip install colorama
```

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

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

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

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

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

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

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

Python in Focus: Stitching Black Holes

One of the biggest scientific, engineering and space-based projects came to a head in 2019, revealing humanity's first glimpse at the universe's most elusive object: a black hole. But what's that got to do with Python?

Imaging a black hole is pretty difficult. The very nature of a black hole means that nothing can escape its immense gravitational field, even light itself. To quote the Wikipedia entry for a black hole:

"A black hole is a region of spacetime exhibiting gravitational acceleration so strong that nothing—no particles or even electromagnetic radiation such as light—can escape from it. The theory of general relativity predicts that a sufficiently compact mass can deform spacetime to form a black hole. The boundary of the region from which no escape is possible is called the event horizon. Although the event horizon has an enormous effect on the fate and circumstances of an object crossing it, no locally detectable features appear to be observed. In many ways, a black hole acts like an ideal black body, as it reflects no light. Moreover, quantum field theory in curved spacetime predicts that event horizons emit Hawking radiation, with the same spectrum as a black body of a temperature inversely proportional to its mass. This temperature is on the order of billionths of a kelvin for black holes of stellar mass, making it essentially impossible to observe."

Not that long ago a black hole was just a collection of theories and mathematics written down on paper, speculated only by the brightest minds of our time. However, as with most things scientific, our understanding of the universe and our abilities to read it have greatly improved and, with the culmination of years of hard work by a collaboration of observatories, scientists and engineers, we got our first image of a black hole.

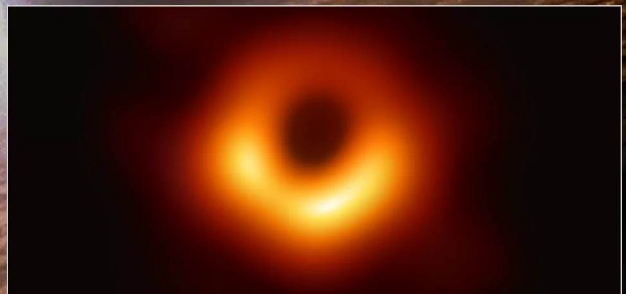
EVENT HORIZON TELESCOPE

One of the problems regarding the imaging of such an object is angular resolution. In Astronomy, the size of the objects in the night sky is referred to by the amount of the sky they take up - units of arc. An arc, or arc second, is a measurement (1/3600 of a degree) that describes the size of an angle in degrees, designated by the symbol °. A full circle is divided into 360° and a right-angle measures 90°. One degree can be divided into 60 arcminutes (abbreviated 60 arcmin or 60'). An arcminute can also be divided into 60 arcseconds (abbreviated 60 arcsec or 60").

For example, looking at the moon, which is roughly 31 arcminutes, imagine drawing a line from you to one side of the moon and another to the opposite side of the moon, the angle between the two lines is the angular size, or angular resolution.

The black hole at the centre of the Messier 87 galaxy, the one that was imaged, is 55 million light years from Earth and has an angular size of 40 microarc seconds, or one millionth of an arcsecond. So, in order to see it, we would need a telescope with a diameter of around 8Km, which simply isn't possible as a single unit.

This is where the Event Horizon Telescope project comes into play. Using a network of eight radio telescopes, scientists were able to take images of the black hole over a period of around six months. Critically timed, using atomic clocks, the telescopes imaged the area of sky containing the black hole and collected the data, swapping from one telescope array to the next as the Earth rotated.



BIG DATA AND PYTHON

This data was then collated across all the telescope arrays to the tune of over a thousand hard drives, which came to an astonishing 5 Petabytes of raw data. The problem now was collating all that data into a workable form and presenting it as an image.

Katie Bouman, a Ph.D. in electrical engineering and computer science from MIT, was pivotal in creating the Python code that was able to stitch all that data together and form the eventual, historic image of a black hole.

Bouman used a number of Python libraries to achieve the result, Numpy, Scipy, Pandas, Jupyter, Matplotlib and Astropy, plus some unique custom Python code – which can be found on Github at **https://github.com/achael/eht-imaging**.

RESULTS

The end result is, of course, the image of the black hole at the centre of the M87 galaxy that's surrounded by a ring of burning gasses. The resolution isn't great, as the team have since admitted, but, as they also state, give it a couple of years and they'll be able to increase the image resolution significantly.

All this is thanks to some clever Python code and some very brilliant scientists and engineers.

```
4   from os import path, makedirs
5
6
7   # set as true if you want to use a model image, false will load the specified image
8   makeModel = True
9   # path to a sample image
10  sampleimage = '../models/rowan_m87.txt'
11
12  # path to array file for loading SEFDs and site locations
13  array = '../arrays/EHT2017_m87.txt'
14
15  # parameters for model image
16  npix = 128
17  fov = 200*eh.RADPERUAS
18  source = 'M87'
19  ra = 19.414182210498385
20  dec = -29.24170032236311
21  zbl = 0.8
22  rf = 230000000000.0
23  mjd = 57854
24
25  ring_radius = 22*eh.RADPERUAS # the radius of the ring
26  ring_width = 10*eh.RADPERUAS # the width of the ring
27  nonun_frac = 0.5 # defines how much brighter the brighter location is on the ring
28  theta_nonun_rad = 270 # defines the angle of the brightest location
29  fracpol = 0.4 # fractional polarization on model image
30  corr = 5 * eh.RADPERUAS # the coherance length of the polarization
31
32  # parameters for simulated data
33  add_th_noise = True # if there are no sefds in obs_orig it will use the sigma for each data point
34  phasecal = False # if False then add random phases to simulate atmosphere
35  ampcal = False # if False then add random gain errors
36  stabilize_scan_phase = True # if true then add a single phase error for each scan to act similar to adhoc phasing
37  stabilize_scan_amp = True # if true then add a single gain error at each scan
38  jones = True # apply jones matrix for including noise in the measurements (including leakage)
39  inv_jones = False # no not invert the jones matrix
40  frcal = True # True if you do not include effects of field rotation
```

Working with Data

Data is everything; it can topple governments, change election results, and tell us the secrets of the universe. Over these coming pages we look at how you can create lists, tuples, dictionaries and multi-dimensional lists, and then how you can use them to forge exciting and useful programs.

In addition, you will learn how you can use the date and time functions, write to les to your system and even create graphical user interfaces that will take your coding skills to new levels and into new project ideas.

Lists

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

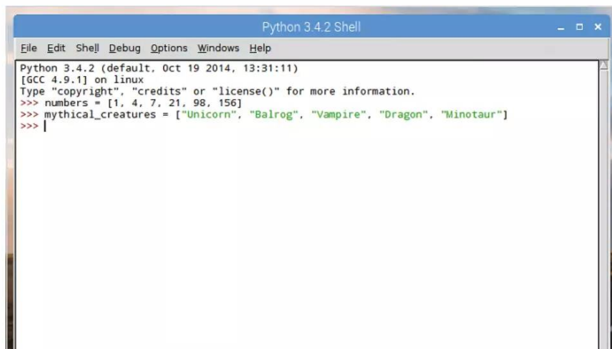
WORKING WITH LISTS

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

STEP 1

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

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



STEP 2

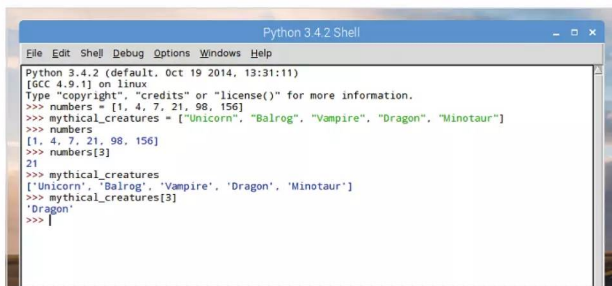
Once you've defined your list you can call each by referencing its name, followed by a number. Lists start the first item entry as 0, followed by 1, 2, 3 and so on. For example:

```
numbers
```

To call up the entire contents of the list.

```
numbers[3]
```

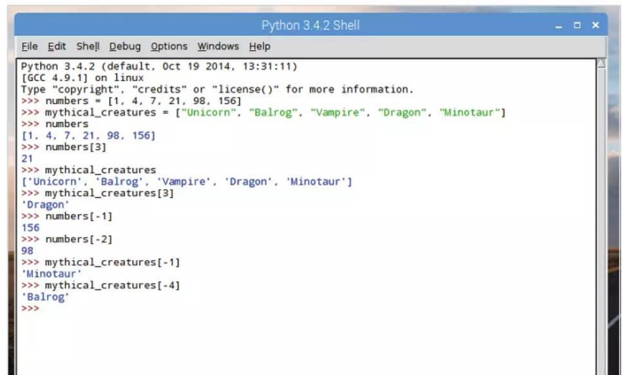
To call the third from zero item in the list (21 in this case).



STEP 3

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

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



STEP 4

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

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

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

STEP 5

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

```
everything = numbers + mythical_creatures
```

Then view the combined list with:

```
everything
```

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> everything = numbers + mythical_creatures
>>> everything
[1, 4, 7, 21, 98, 156, 'Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>>
```

STEP 6

Items can be added to a list by entering:

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

Or for strings:

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

Or by using the append function:

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

```
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> |
```

STEP 7

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

```
del numbers[7]
```

Alternatively, by item name:

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

```
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> del numbers[7]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures.remove("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> |
```

STEP 8

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

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

To remove it:

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

```
Type "copyright", "credits" or "license()" for more information.
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__'
, '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__'
, '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers.insert(4, 62)
>>> numbers
[1, 4, 7, 21, 62, 98, 156]
#
>>> numbers.pop(4)
62
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> |
```

STEP 9

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

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

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list("David")
['D', 'a', 'v', 'i', 'd']
>>> name=list("David Hayward")
>>> name
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd']
>>> age=[44]
>>> user = name + age
>>> user
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd', 44]
>>> |
```

STEP 10

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

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? Conan of Cimmeria
How old are you: 44
>>> user
['C', 'o', 'n', 'a', 'n', ' ', 'o', 'f', ' ', 'C', 'i', 'm', 'm', 'e', 'r', 'i',
 'a', 44]
>>>
```

```
namelist.py - /home/pi/Docu
File Edit Format Run Options Windows
name=input("What's your name? ")
lname=list(name)
age=int(input("How old are you: "))
lage=[age]

user = lname + lage
```

Tuples

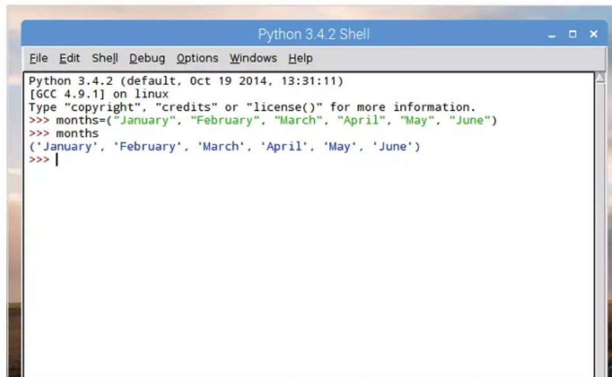
Tuples are very much identical to lists. However, where lists can be updated, deleted or changed in some way, a tuple remains a constant. This is called immutable and they're perfect for storing fixed data items.

THE IMMUTABLE TUPLE

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

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

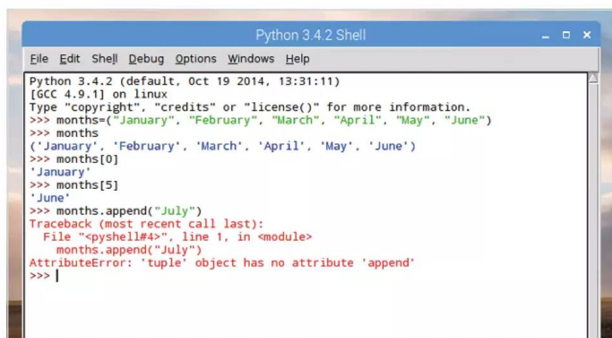
```
months=("January", "February", "March", "April",
"May", "June")
months
```



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

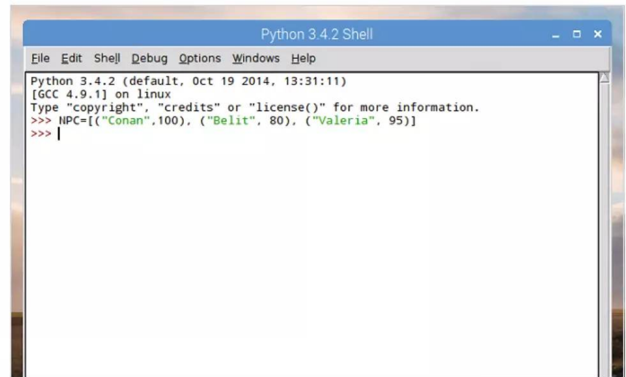
```
months[0]
months[5]
```

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



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

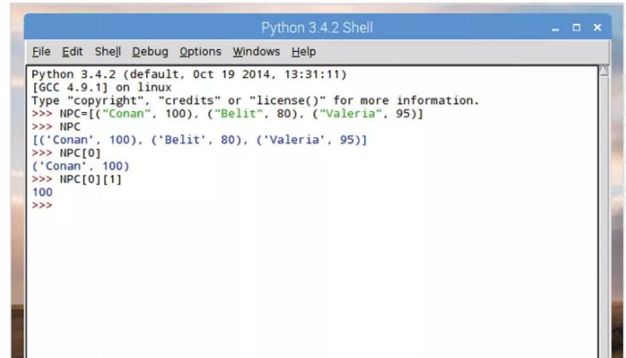
```
NPC=[("Conan", 100), ("Belit", 80), ("Valeria",
95)]
```



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

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

Will display 100.



STEP 5

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

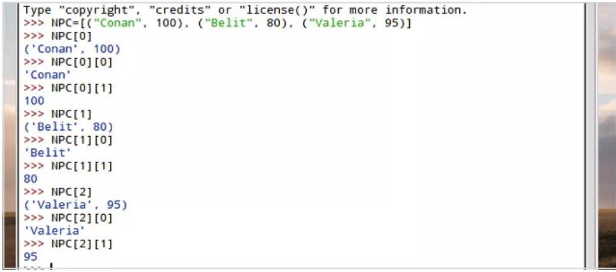
```
NPC[2][1]
```

STEP 6

This means of course that the indexing follows thus:

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

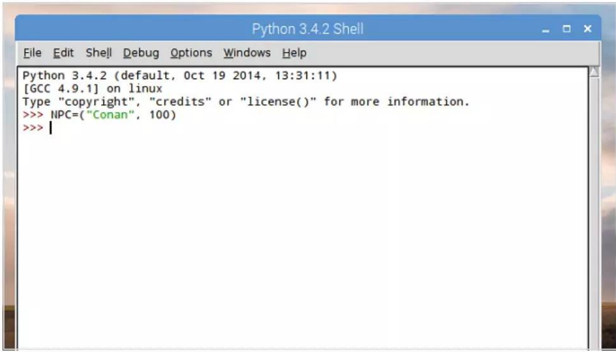
Which as you can imagine, gets a little confusing when you've got a lot of tuple data to deal with.



STEP 7

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

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

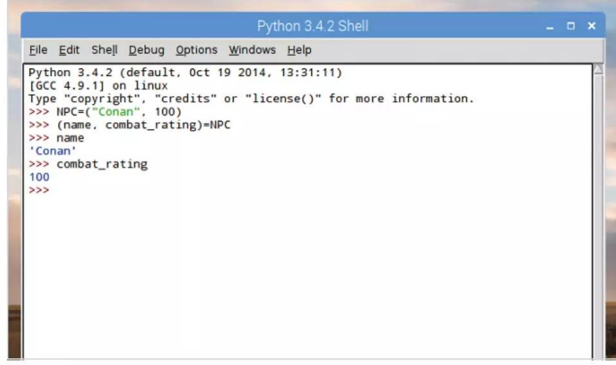


STEP 8

Now unpack the tuple into two corresponding variables:

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

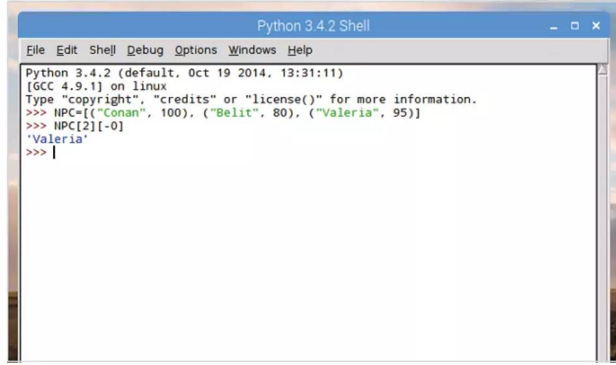
You can now check the values by entering name and combat_rating.



STEP 9

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

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



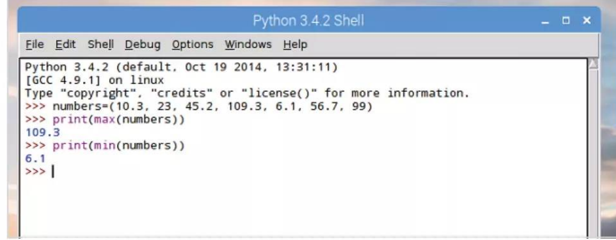
STEP 10

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

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

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

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



Dictionaries

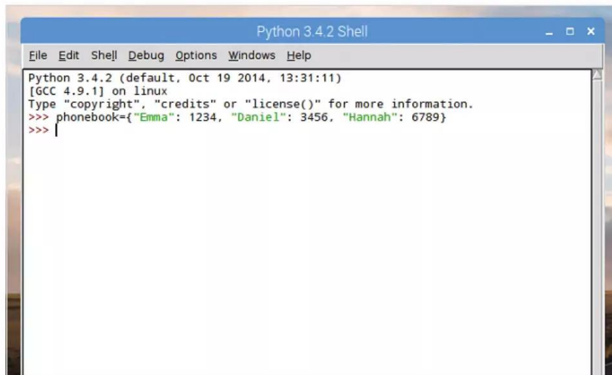
Lists are extremely useful but dictionaries in Python are by far the more technical way of dealing with data items. They can be tricky to get to grips with at first but you'll soon be able to apply them to your own code.

KEY PAIRS

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

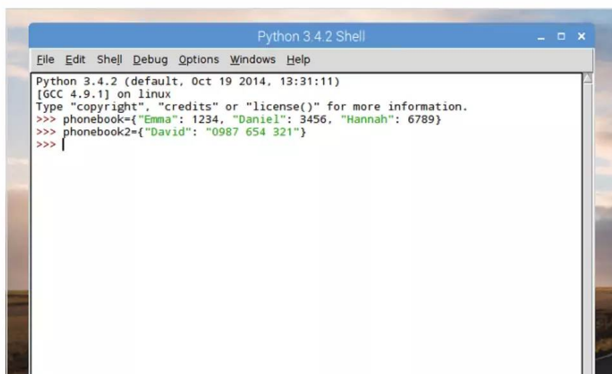
STEP 1 Let's say you want to create a phonebook in Python. You would create the dictionary name and enter the data in curly brackets, separating the key and value by a colon `Key:Value`. For example:

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

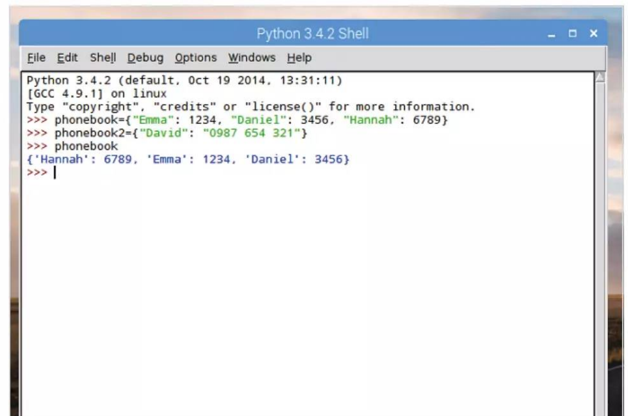


STEP 2 Just as with most lists, tuples and so on, strings need be enclosed in quotes (single or double), whilst integers can be left open. Remember that the value can be either a string or an integer, you just need to enclose the relevant one in quotes:

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

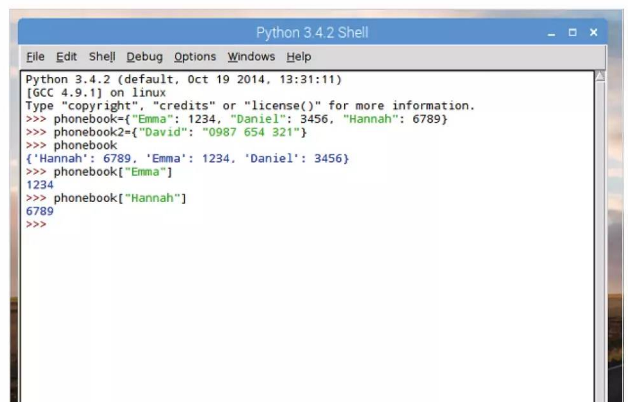


STEP 3 As with lists and tuples, you can check the contents of a dictionary by giving the dictionary a name: phonebook, in this example. This will display the data items you've entered in a similar fashion to a list, which you're no doubt familiar with by now.



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

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



STEP 5

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>>
```

STEP 6

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>> del phonebook["David"]
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>>
```

STEP 7

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

```
phonebook={}
```

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
phonebook={}
```

STEP 8

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

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

```
phonebook={}

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

STEP 9

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

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

```
phonebook={}

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

phonebook[name] = number
```

STEP 10

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

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

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

Splitting and Joining Strings

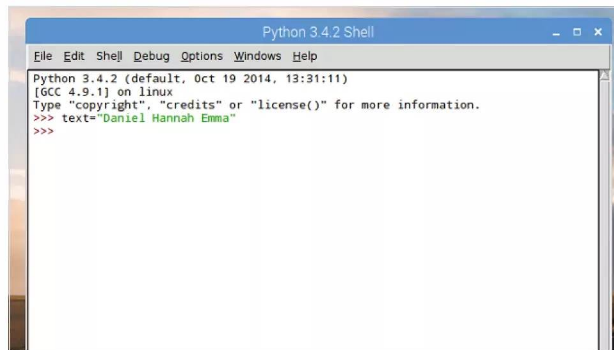
When dealing with data in Python, especially from a user's input, you will undoubtedly come across long sets of strings. A useful skill to learn in Python programming is being able to split those long strings for better readability.

STRING THEORIES

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

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

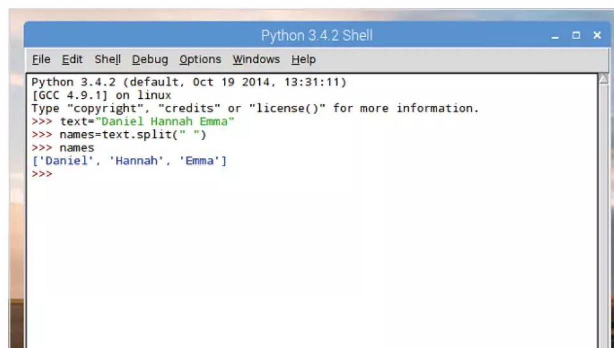
```
text="Daniel Hannah Emma"
```



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

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

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



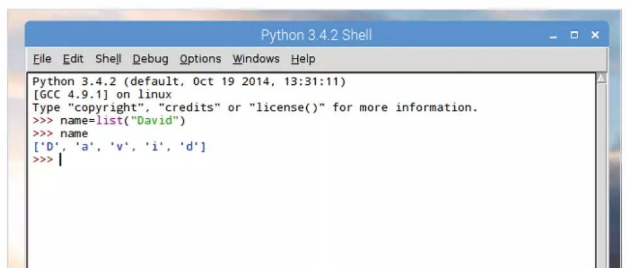
STEP 3 Note that the text.split part has the brackets, quotes, then a space followed by closing quotes and brackets. The space is the separator, indicating that each list item entry is separated by a space. Likewise, CSV (Comma Separated Value) content has a comma, so you'd use:

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

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

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

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



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

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

This will display 'abcde' in the Shell.

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> alphabet="".join(["a","b","c","d","e"])
>>> alphabet
'abcde'
>>>
```

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

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name=list("David")
>>> name
['D', 'a', 'v', 'i', 'd']
>>> name="".join(name)
>>> name
'David'
>>>
```

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

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>>
```

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>> colours=["Red", "Green", "Blue"]
>>> col=",".join(colours)
>>> col
'Red,Green,Blue'
>>>
```

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> title="conan the cimmerian"
>>> title.capitalize()
'Conan the cimmerian'
>>> title.title()
'Conan The Cimmerian'
>>>
```

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

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

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

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> message="Have a nice day"
>>> "nice" in message
True
>>> "bad" not in message
True
>>> "day" not in message
False
>>> "night" in message
False
>>>
```

Formatting Strings

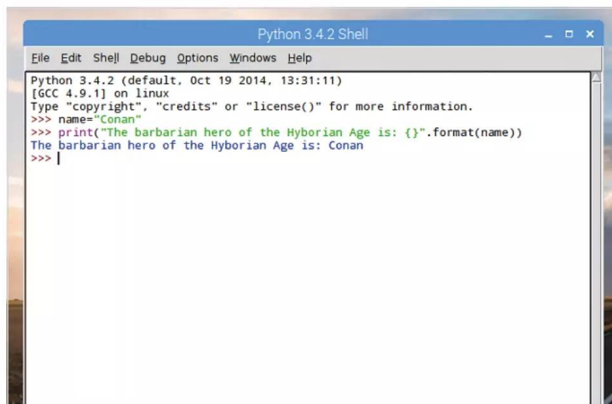
When you work with data, creating lists, dictionaries and objects you may often want to print out the results. Merging strings with data is easy especially with Python 3, as earlier versions of Python tended to complicate matters.

STRING FORMATTING

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

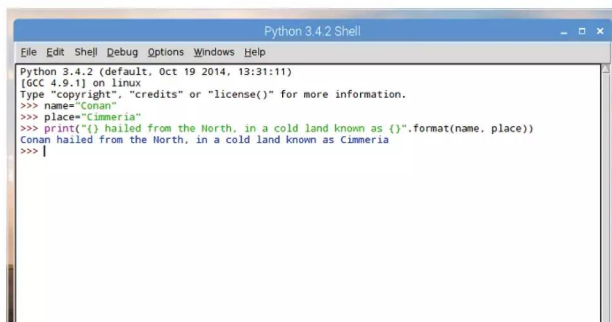
STEP 1 The basic formatting in Python is to call each variable into the string using the curly brackets:

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



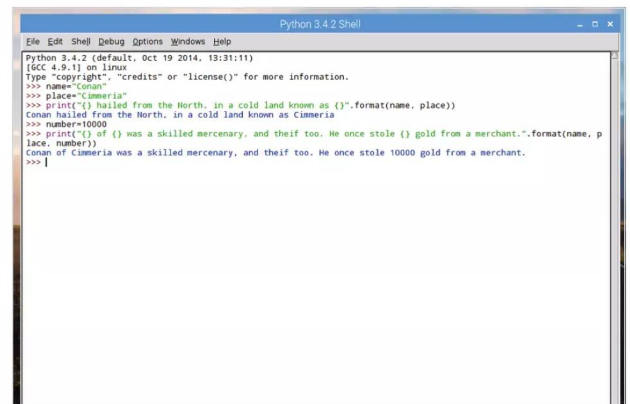
STEP 2 Remember to close the print function with two sets of brackets, as you've encased the variable in one, and the print function in another. You can include multiple cases of string formatting in a single print function:

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

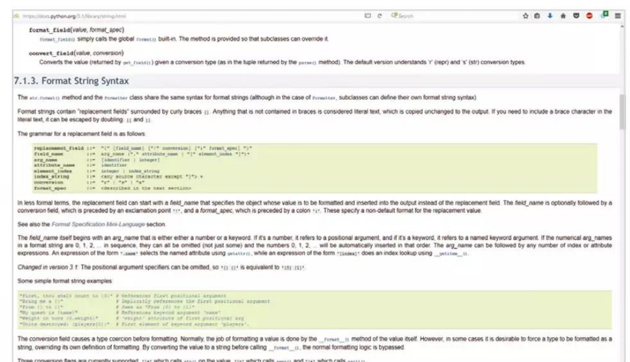


STEP 3 You can of course also include integers into the mix:

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

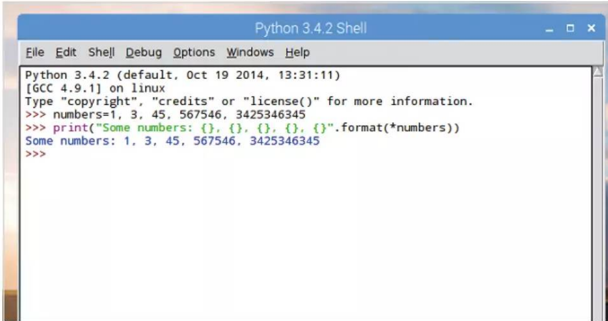


STEP 4 There are many different ways to apply string formatting, some are quite simple, as we've shown you here; others can be significantly more complex. It all depends on what you want from your program. A good place to reference frequently regarding string formatting is the Python Docs webpage, found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



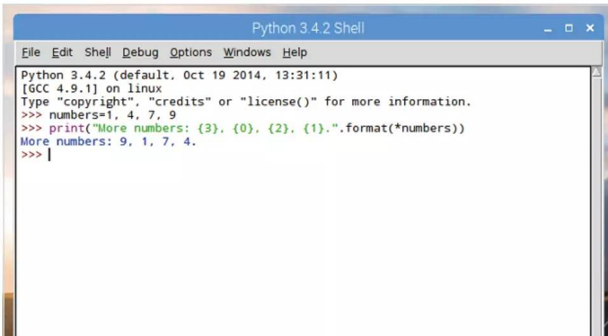
STEP 5 Interestingly you can reference a list using the string formatting function. You need to place an asterisk in front of the list name:

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



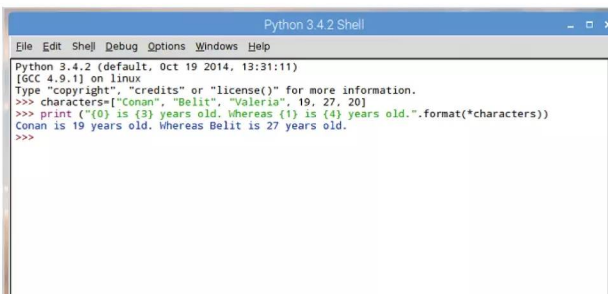
STEP 6 With indexing in lists, the same applies to calling a list using string formatting. You can index each item according to its position (from 0 to however many are present):

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



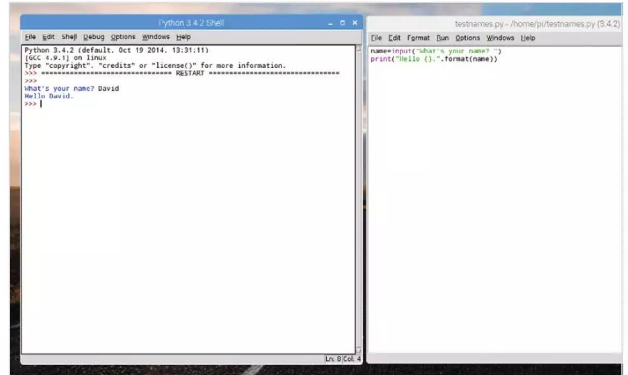
STEP 7 And as you probably suspect, you can mix strings and integers in a single list to be called in the .format function:

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



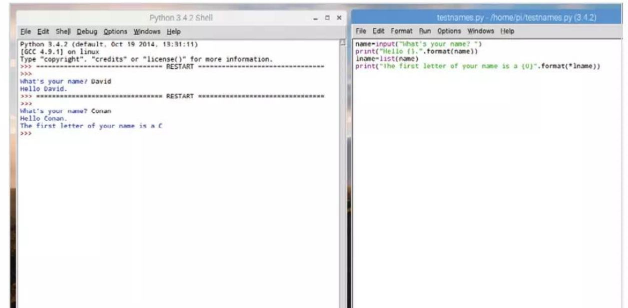
STEP 8 You can also print out the content of a user's input in the same fashion:

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



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

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

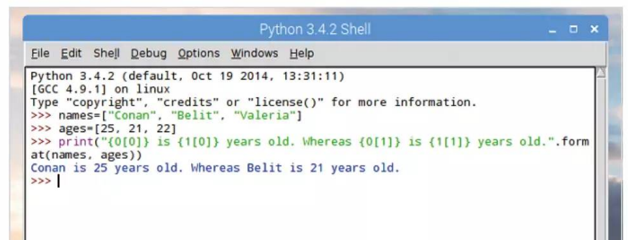


STEP 10 You can also call upon a pair of lists and reference them individually within the same print function. Looking back the code from Step 7, you can alter it with:

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

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

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



Date and Time

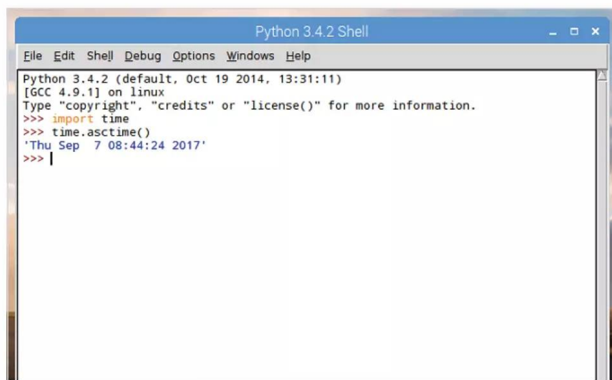
When working with data it's often handy to have access to the time. For example, you may want to time-stamp an entry or see at what time a user logged into the system and for how long. Luckily acquiring the date and time is easy, thanks to the Time module.

TIME LORDS

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

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

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



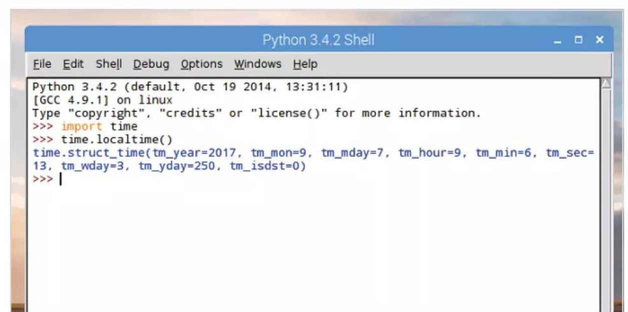
STEP 2 The time function is split into nine tuples, these are divided up into indexed items, as with any other tuple, and shown in the screen shot below.

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

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

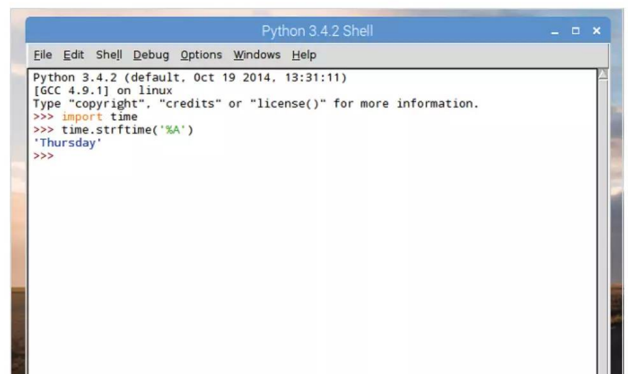
```
time.local.time()
```

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



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

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



STEP 5

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

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

```
Python 3.4.2 Shell
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>>
```

STEP 6

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

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

```
Python 3.4.2 Shell
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>> time.strftime("%H:%M")
'09:43'
>>>
```

STEP 7

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

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

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

STEP 8

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

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

Then there's:

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

```
logintime.py - /home/pi/Documents/Python Code/logintime.py (3.4.2)
import time

start_time=time.time()
name=input("Enter login name: ")
endtime=time.time()-start_time

print("Welcome", name, "\d")

print("User:", name, "logged in at", time.strftime("%H:%M"))
print ("It took", name, endtime, "to login to their account.")
```

STEP 9

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

```
Python 3.4.2 Shell
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter login name: David
Welcome David \d
User: David logged in at 09:52
It took David 5.311823129653931 to login to their account.
>>>
```

STEP 10

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

```
Python 3.4.2 Shell
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> help(time)
Help on built-in module time:

NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).

    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
```

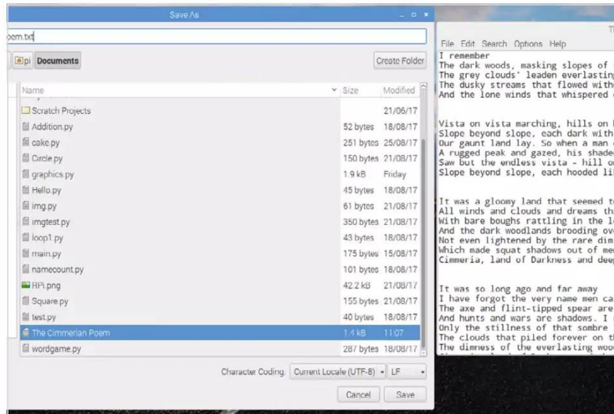
Opening Files

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

OPEN, READ AND WRITE

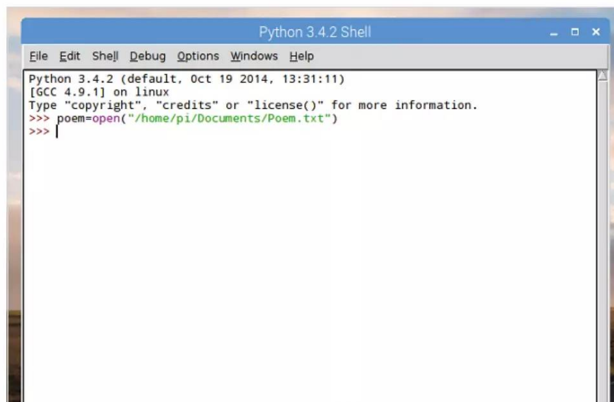
In Python you create a file object, similar to creating a variable, only pass in the file using the open() function. Files are usually categorised as text or binary.

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



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

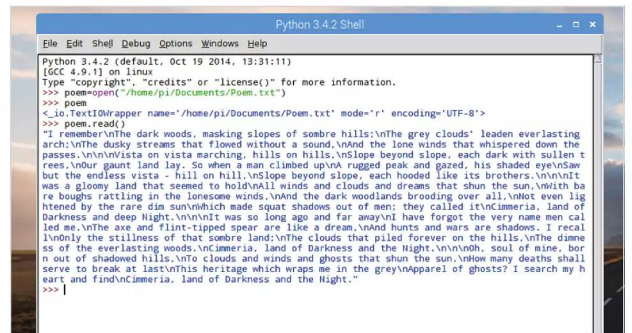
```
poem=open("/home/pi/Documents/Poem.txt")
```



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

```
poem.read()
```

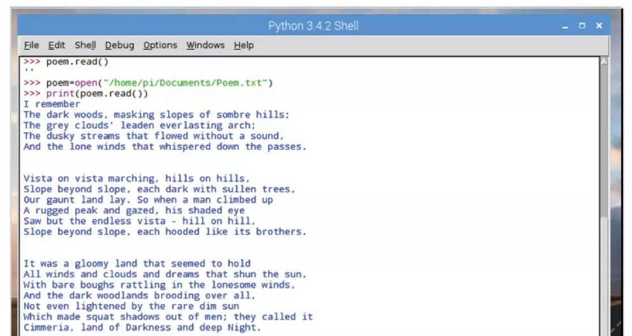
Note than a /n entry in the text represents a new line, as you used previously.



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

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

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



STEP 5

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

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

Displays the first five characters, whilst again entering:

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.read(5)
'I rem'
>>> poem.read(5)
'ember'
>>> |
```

STEP 6

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

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

Will display the first line of the text with:

```
poem.readline()
```

Displaying the next line of text once more.

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.readline()
'I remember\n'
>>> poem.readline()
'The dark woods, masking slopes of sombre hills;\n'
>>> |
```

STEP 7

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> line=poem.readline()
>>> line
'I remember\n'
>>>|
```

STEP 8

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> lines=poem.readlines()
>>> lines[0]
'I remember\n'
>>> lines[1]
'The dark woods, masking slopes of sombre hills;\n'
>>> lines[2]
"The grey clouds' leaden everlasting arch;\n"
>>>
```

STEP 9

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

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

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> for lines in poem:
        print(lines)


I remember

The dark woods, masking slopes of sombre hills;

The grey clouds' leaden everlasting arch;
```

STEP 10

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

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

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

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.
```

```
readpoem.py - /home/pi/Documents/Python Code/readpoem.py (3.4.2)
File Edit Format Run Options Windows Help
import time

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

lines=poem.read()

for lines in lines:
    print(lines, end="")
    time.sleep(.15)
```

Writing to Files

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

WRITE AND CLOSE

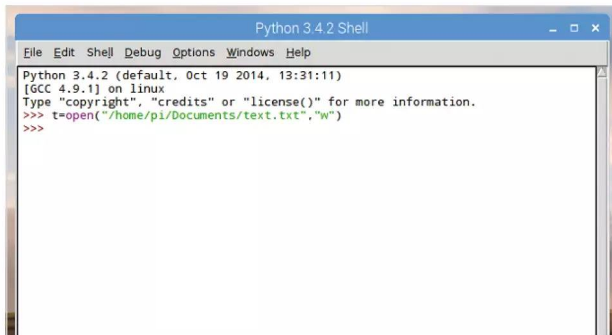
The write() function is slightly more complex than read(). Along with the filename you must also include an access mode which determines whether the file in question is in read or write mode.

STEP 1

Start by opening IDLE and enter the following:

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

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

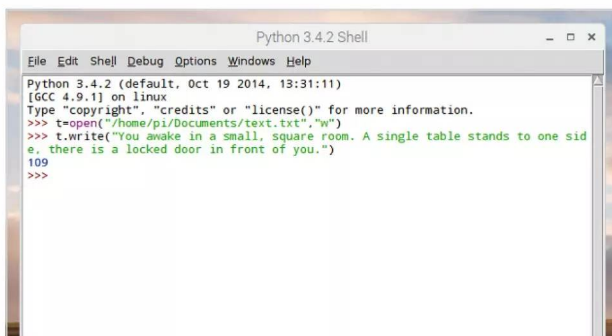


STEP 2

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

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

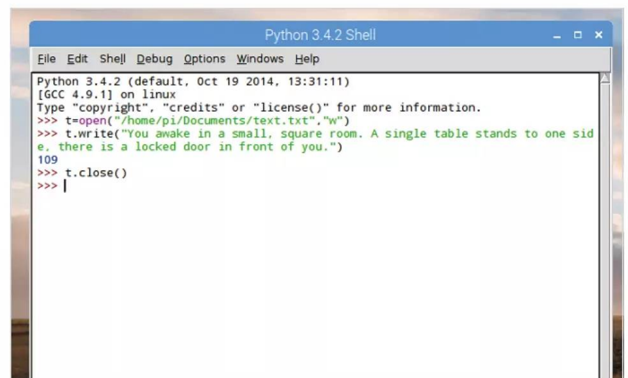
Note the 109. It's the number of characters you've entered.



STEP 3

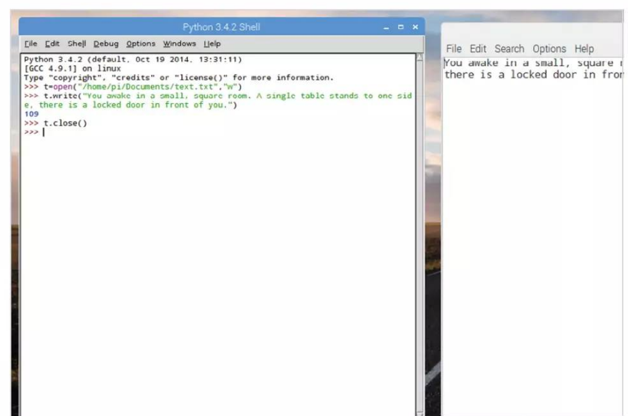
However, the actual text file is still blank (you can check by opening it up). This is because you've written the line of text to the file object but not committed it to the file itself. Part of the write() function is that you need to commit the changes to the file; you can do this by entering:

```
t.close()
```



STEP 4

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



STEP 5

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

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

STEP 6

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

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

STEP 7

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

STEP 8

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

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

STEP 9

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

```
pi=math.pi
```

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

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

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

STEP 10

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

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

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

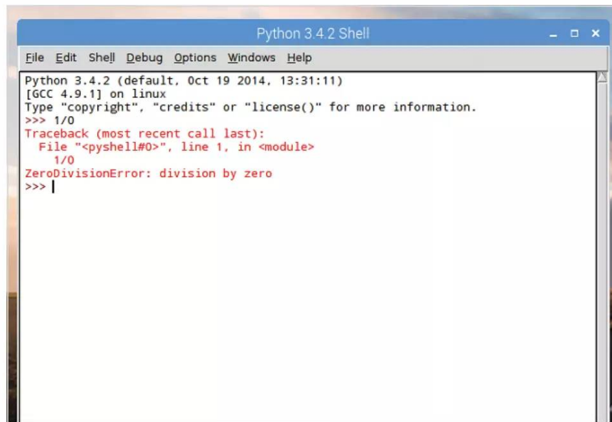
Exceptions

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

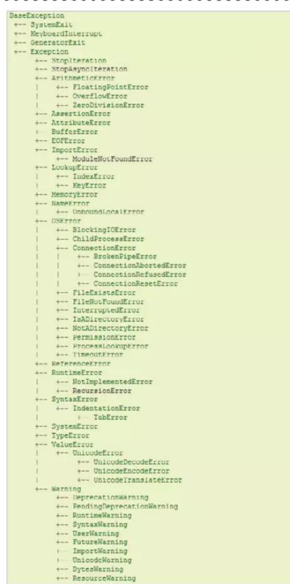
EXCEPTIONAL OBJECTS

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

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

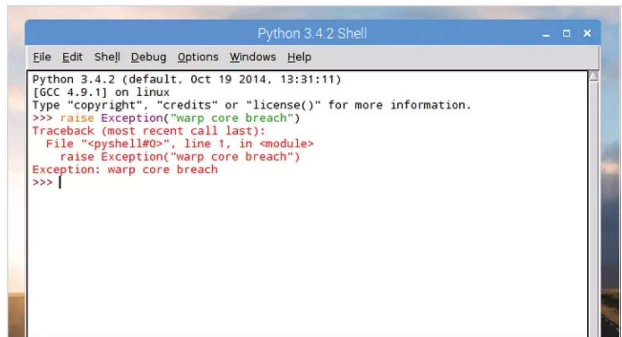


STEP 2 Most exceptions are raised automatically when Python comes across something that's inherently wrong with the code. However, you can create your own exceptions that are designed to contain the potential error and react to it, as opposed to letting the code fail.

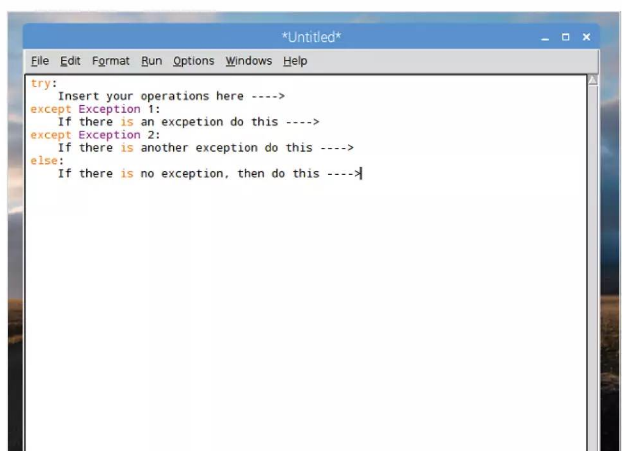


STEP 3 You can use the functions raise exception to create our own error handling code within Python. Let's assume your code has you warping around the cosmos, too much however results in a warp core breach. To stop the game from exiting due to the warp core going supernova, you can create a custom exception:

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

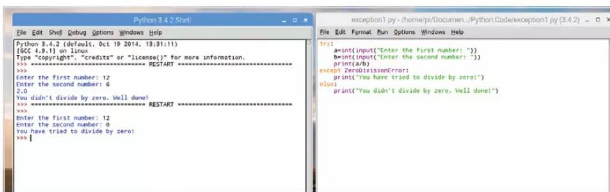


STEP 4 To trap any errors in the code you can encase the potential error within a try: block. This block consists of try, except, else, where the code is held within try:, then if there's an exception do something, else do something else.



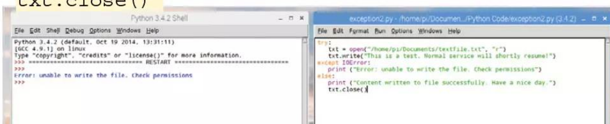
STEP 5 For example, use the divide by zero error. You can create an exception where the code can handle the error without Python quitting due to the problem:

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

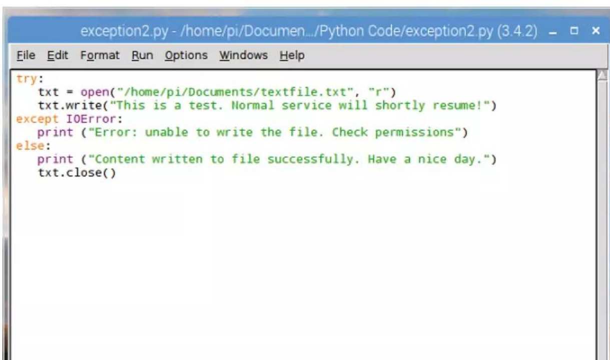


STEP 6 You can use exceptions to handle a variety of useful tasks. Using an example from our previous tutorials, let's assume you want to open a file and write to it:

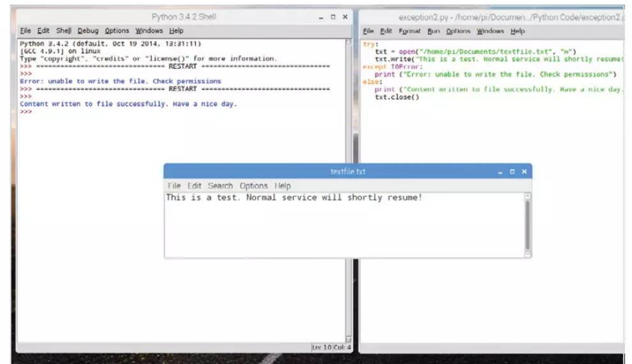
```
try:
 txt = open("/home/pi/Documents/textfile.txt",
"r")
 txt.write("This is a test. Normal service will
shortly resume!")
except IOError:
 print ("Error: unable to write the file. Check
permissions")
else:
 print ("Content written to file successfully.
Have a nice day.")
 txt.close()
```



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

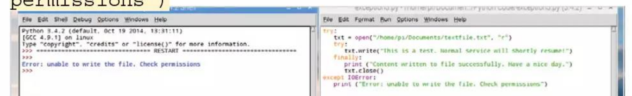


STEP 8 Naturally, you can quickly fix the issue by changing the "r" read only instance with a "w" for write. This, as you already know, will create the file and write the content then commit the changes to the file. The end result will report a different set of circumstances, in this case, a successful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

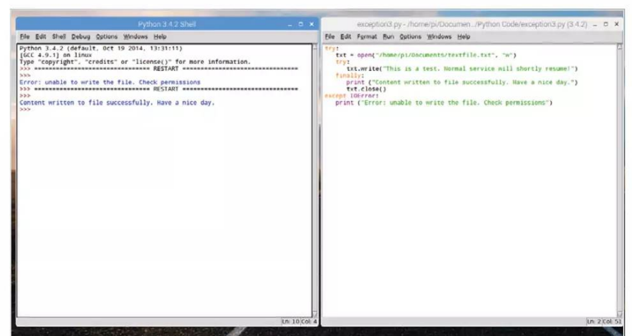


STEP 9 You can also use a finally: block, which works in a similar fashion but you can't use else with it. To use our example from Step 6:

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



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



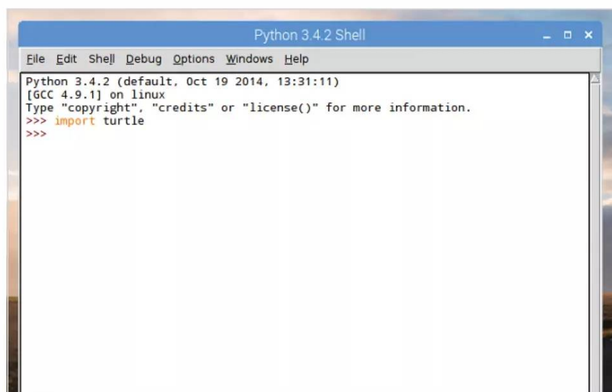
Python Graphics

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

GOING GRAPHICAL

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

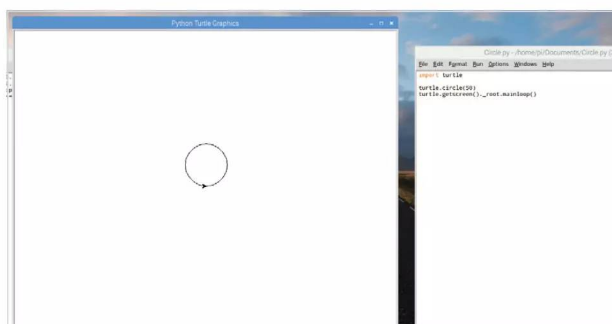
STEP 1 One of the best graphical modules to begin learning Python graphics is Turtle. The Turtle module is, as the name suggests, based on the turtle robots used in many schools, that can be programmed to draw something on a large piece of paper on the floor. The Turtle module can be imported with: `import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



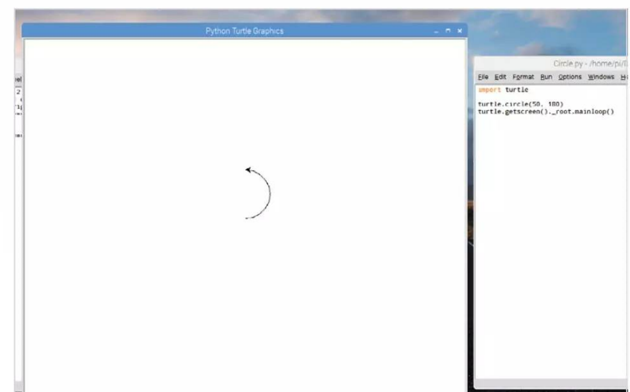
STEP 2 Let's begin by drawing a simple circle. Start a New File, then enter the following code:

```
import turtle

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

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



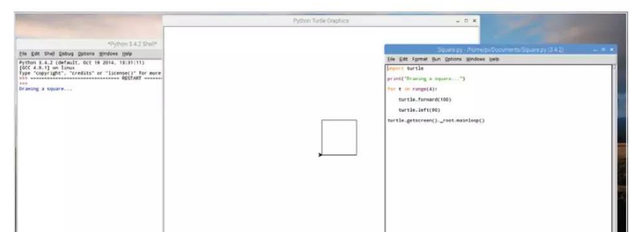
STEP 3 The command turtle.circle(50) is what draws the circle on the screen, with 50 being the size. You can play around with the sizes if you like, going up to 100, 150 and beyond; you can draw an arc by entering: `turtle.circle(50, 180)`, where the size is 50, but you're telling Python to only draw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



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

```
import turtle

print("Drawing a square…")

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

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



STEP 5

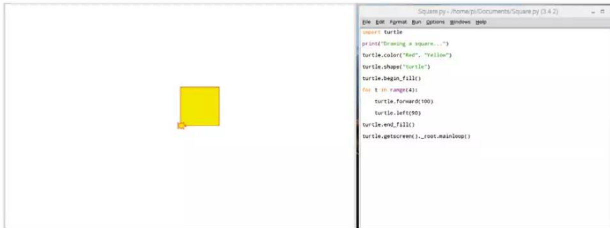
You can add a new line to the square code to add some colour:

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

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

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

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

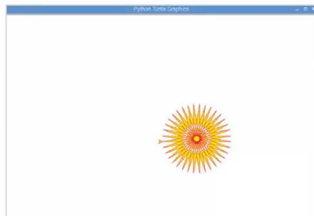


STEP 6

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

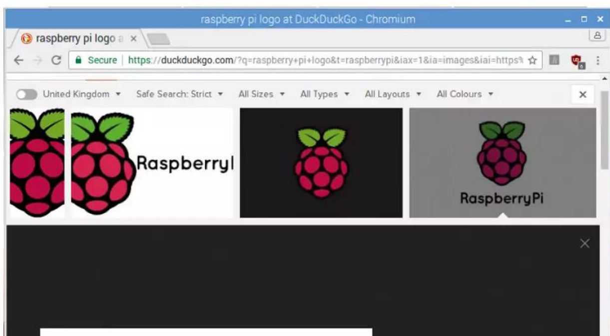
```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
  forward(200)
  left(170)
  if abs(pos()) < 1:
    break
end_fill()
done()
```

It's a different method, but very effective.



STEP 7

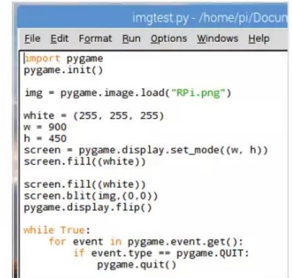
Another way in which you can display graphics is by using the Pygame module. There are numerous ways in which pygame can help you output graphics to the screen but for now let's look at displaying a predefined image. Start by opening a browser and finding an image, then save it to the folder where you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



STEP 8

Now let's get the code by importing the Pygame module:

```
import pygame
pygame.init()

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

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

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

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



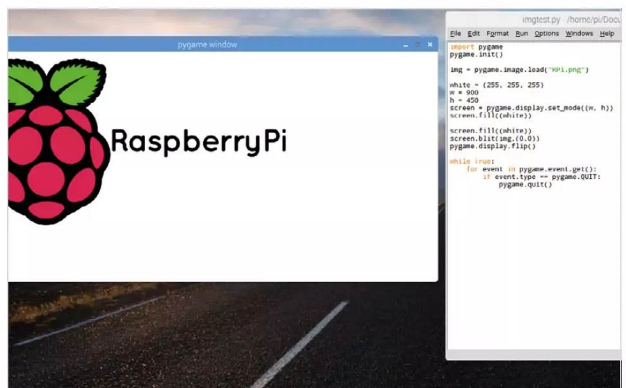
STEP 9

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

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

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

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

STEP 10

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



Combining What You Know So Far

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

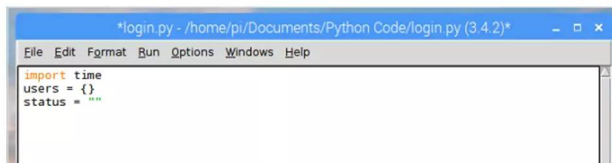
LOGGING IN

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

STEP 1

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

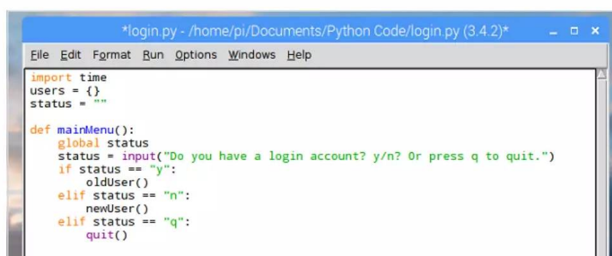
```
import time
users = {}
status = ""
```



STEP 2

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

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



STEP 3

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

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

STEP 4

The newUser function is next:

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

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

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

STEP 5

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

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

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

STEP 6

Now to create the oldUser function:

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

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

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

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

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

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

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

STEP 7

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

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

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

STEP 8

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

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

login.py - /home/pi/Documents/Python Code/login.py (3.4.2)

File Edit Format Run Options Windows Help

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

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

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

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

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

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

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

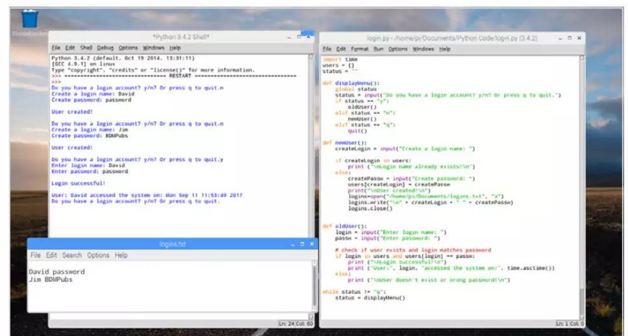
STEP 9

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

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

STEP 10

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



Python in Focus: Gaming

Although not always considered as the ideal programming language for developing games, Python has come a long way in recent years and is now one of the contributing elements to a huge number of titles.

The video game industry generates something in the region of $140 billion each year, and that number is growing fast. It's a long way from the 8-bit days of the Commodore 64 and ZX Spectrum; the arcade titles that used to devour our pocket money and the wood panelled home consoles that Atari lovingly developed. These days, it's all about teams of coders, graphic artists, musicians, PR, projects and development platforms.

GAME CODE

Coding a game from scratch, using raw code, has become something of the past. Most games these days are created using a range of development tools. These tools can be off-the-shelf engines, such as the Unreal Engine, while others are custom built around an original product, such as the world generating engine that Bethesda use for the Skyrim and Fallout series of games. Others examples can be coded from the ground up, but these are generally few and far between. So where does Python fit into all this?

The limiting factor with Python is performance. While most games require a huge degree of performance from the platform for which they are written, Python's code, which is good, isn't really designed to cope with the fast-paced formula on which games such as Battlefield or the Call of Duty series are based. These games are often coded with C++, or some other form of low-level programming language. But that doesn't mean Python is left out in the cold when it comes to game development, in fact it's quite the opposite.

```
MONITOR FOR 6802 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
```

BUILDING TOOLS

In the game industry, Python is mostly limited to the development of in-game tools used by the developers of the game, or to help bridge the gaps between different areas of code. For example, in-game tools coded in Python can be used by designers to create levels for the game, or specific elements that would make up a character's inventory, or even creating dialog between the player and non-playing characters in the game.

You will also find that Python can be used to control the game's AI (Artificial Intelligence), which will give the characters in a game a certain element of life. As an example, the popular Sims games consist of characters other than the one the gamer controls. These Sims will go about their business with their actions determined by the player's choices, this involves an advanced form of Artificial Intelligence that is coded using Python.

Other examples include many of the available open world games, where the introduction of the player will change the course of a village's, Town's, or even city's inhabitant's behaviour. Blow up a few cars in the middle of the street and it'll affect the way the other drivers behave; jump up and down on top of a market stall in the middle of a medieval village and the folk around you will react. This, again, is all down to Python code written within the main code of the game, alongside the game development engine.

PYTHON-POWERED GAMES

Some good examples of the types of games in which Python is used are the following:

Battlefield 2 – Python is used for the game's add-ons and functionality of the player elements.

The Sims – AI, and many of the game's interactions.

Civilisation – Python is used throughout the Civ games, controlling movement and the non-player AI.

Eve Online – Utilises Python for floating point number calculations and other tasks.

World of Tanks – Python is used to control AI objects and detail the large amount of graphical data.

In particular, it's worth noting that Python's use in games is due to its ability to automate repetitive tasks quickly. While another programming language may be faster at drawing the graphics on the screen, Python can quickly repeat resizing hundreds of textures in batches. There's also Python's excellent and sizeable libraries that can be tweaked for certain tasks, specifically in-game tasks freeing up other components to deliver the performance that modern games need.

KEEP ON GAMING

In short, while Python may not be the ideal language with which to create a modern game entirely, its use is often behind the scenes, in areas where other programming languages will struggle. Python can be used as the glue that sticks elements of game technologies together, creating complex AI or simply designing a dialog box.

Glossary of Python Terms

Just like most technology, Python contains many confusing words and acronyms. Here then, for your own sanity, is a handy glossary to help you keep on top of what's being said when the conversation turns to Python programming.

Argument
The detailed extra information used by Python to perform more detailed commands. Can also be used in the command prompt to specify a certain runtime event.

Block
Used to describe a section or sections of code that are grouped together.

Break
A command that can be used to exit a for or while loop. For example, if a key is pressed to quit the program, Break will exit the loop.

Class
A class provides a means of bundling data and functionality together. They are used to encapsulate variables and functions into a single entity.

Comments
A comment is a section of real world wording inserted by the programmer to help document what's going on in the code. They can be single line or multi-line and are defined by a # or '''.

Debian
A Linux-based distro or distribution that forms the Debian Project. This environment offers the user a friendly and stable GUI to interact with along with Terminal commands and other forms of system level administration.

Def
Used to define a function or method in Python.

Dictionaries
A dictionary in Python is a data structure that consists of key and value pairs.

Distro
Also Distribution, an operating system that uses the Linux Kernel as its core but offers something different in its presentation to the end user.

Editor
An individual program, or a part of the graphical version of Python, that enables the user to enter code ready for execution.

Exceptions
Used as a means of breaking from the normal flow of a code block in order to handle any potential errors or exceptional conditions within the program.

Expression
Essentially, Python code that produces a value of something.

Float
An immutable floating point number used in Python.

Function
Used in Python to define a sequence of statements that can be called or referenced at any time by the programmer.

GitHub
A web-based version control and collaboration portal designed for software developers to better manage source code.

Global Variable
A variable that is useable anywhere in the program.

Graphics
The use of visual interaction with a program, game or operating system. Designed to make it easier for the user to manage the program in question.

GUI
Graphical User Interface. The interface which most modern operating systems use to enable the user to interact with the core programming of the system. A friendly, easy to use graphical desktop environment.

High-Level Language
A programming language that's designed to be easy for people to read.

IDLE
Stands for Integrated Development Environment or Integrated Development and Learning Environment.

Immutable
Something that cannot be changed after it is created.

Import
Used in Python to include modules together with all the accompanying code, functions and variables they contain.

Indentation
Python uses indentation to delimit blocks of code. The indents are four spaces apart, and are often created automatically after a colon is used in the code.

Integer
A number data type that must be a whole number and not a decimal.

Interactive Shell
The Python Shell, which is displayed whenever you launch the graphical version of Python.

Kernel
The core of an operating system, which handles data processing, memory allocation, input and output, and processes information between the hardware and programs.

Linux
An open source operating system that's modelled on UNIX. Developed in 1991 by Finnish student Linus Torvalds.

Lists
A Python data type that contains collections of values, which can be of any type and can readily be modified.

Local Variable
A variable that's defined inside a function and is only useable inside that function.

Loop
A piece of code that repeats itself until a certain condition is met. Loops can encase the entire code or just sections of it.

Module
A Python file that contains various functions that can be used within another program to further extend the effectiveness of the code.

Operating System
Also OS. The program that's loaded into the computer after the initial boot sequence has completed. The OS manages all the other programs, graphical user interface (GUI), input and output and physical hardware interactions with the user.

Output
Data that is sent from the program to a screen, printer or other external peripheral.

PIP
Pip Installs Packages. A package management system used to install and manage modules and other software written in Python.

Print
A function used to display the output of something to the screen.

Prompt
The element of Python, or the Command Line, where the user enters their commands. In Python it's represented as >>> in the interactive shell.

Pygame
A Python module that's designed for writing games. It includes graphics and sound libraries and was first developed in October 2000.

Python
An awesome programming language that's easy to learn and use, whilst still being powerful enough to enjoy.

Random
A Python module that implements a pseudo-random character generator using the Mersenne Twister PRNG.

Range
A function that used to return a list of integers, defined by the arguments passed through it.

Root
The bottom level user account used by the system itself. Root is the overall system administrator and can go anywhere, and do anything, on the system.

Sets
Sets are a collection of unordered but unique data types.

Strings
Strings can store characters that can be modified. The contents of a string are alphanumerical and can be enclosed by either single or double quote marks.

Terminal
Also Console or Shell. The command line interface to the operating system, namely Linux, but also available in macOS. From there you can execute code and navigate the filesystem.

Tkinter
A Python module designed to interact with the graphical environment, specifically the tk-GUI (Tool Kit Graphical User Interface).

Try
A try block allows exceptions to be raised, so any errors can be caught and handled according to the programmer's instructions.

Tuples
An immutable Python data type that contains an ordered set of either letters or numbers.

UNIX
A multitasking, multiuser operating system designed in the '70s at the Bell Labs Research Centre. Written in C and assembly language

Variables
A data item that has been assigned a storage location in the computer's memory.

X
Also X11 or X-windows. The graphical desktop used in Linux-based systems, combining visual enhancements and tools to manage the core operating system.

Zen of Python
When you enter: `import this` into the IDLE, the Zen of Python is displayed.

Save a whopping 25% Off!

ALL Tech Manuals

with

Not only can you learn new skills and master your tech, but you can now SAVE 25% off all of our coding and consumer tech digital and print guidebooks!

Simply use the following exclusive code at checkout:

NYHF23CN

www.pclpublications.com

Python For Beginners

20 - ISBN: 978-1-912847-11-2

Published by: Papercut Limited

Digital distribution by: Readly AB

© 2024 Papercut Limited All rights reserved. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Papercut Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the tech industry, the publisher cannot guarantee that all apps and software will work on every version of device. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Any app images reproduced on the front cover are solely for design purposes and are not representative of content. We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the reviewer - as an individual - and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion and content. This is an independent publication and as such does not necessarily reflect the views or opinions of the producers of apps or products contained within. This publication is 100% unofficial. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of brands and products. Additional images contained within this publication are reproduced under licence from Shutterstock. Prices, international availability, ratings, titles and content are subject to change. All information was correct at time of publication. Some content may have been previously published in other volumes or titles.

Papercut Limited
Registered in England & Wales No: 04308513

ADVERTISING – For our latest media packs please contact:
Brad Francis - brad@papercutltd.co.uk
Web - www.pclpublications.com

INTERNATIONAL LICENSING – Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: james@papercutltd.co.uk