

TRICKS  TIPS

100% INDEPENDENT

Raspberry **PI**

OVER
475
SECRETS &
HACKS



We share our
awesome tips
and shortcuts.

Next level
secrets, hacks
and fixes.

Advanced
guides
and tutorials



Save a Huge 50% Off ALL Digital Manuals!*



As a token of our thanks for your support,  **Papercut** bring you this unmissable offer! Not only can you learn new skills and master your tech, but you can now **SAVE 50%** off all of our coding, photography and consumer tech digital manuals!

To take advantage of this amazing offer visit:

www.pclpublications.com/product-category/digital/

Make your selection(s) and then simply use the following exclusive code at checkout to get your purchase at half price:

SAVE50

*Please note this exclusive single use per user discount code requires sign up for activation. Discount cannot be applied against discounted items, print titles or used with an additional discount code.



Raspberry Pi

Raspberry Pi Tricks & Tips is the perfect digital publication for the user that wants to take their skill set to the next level. Do you want to enhance your user experience? Or wish to gain insider knowledge? Do you want to learn directly from experts in their field? Learn the numerous short cuts that the professionals use? Over the pages of the new advanced user guide you will learn everything you will need to know to become a more confident, better skilled and experienced owner. A user that will make the absolute most of their Raspberry Pi use and ultimately their coding and digital creation themselves. An achievement you can earn by simply enabling us to exclusively help and teach you the abilities we have gained over our decades of experience.

*Over the page
our journey continues,
and we will be with you
at every stage to advise,
inform and ultimately
inspire you to
go further.*

Contents

6 Python on the Pi

- | | | | |
|----|--------------------------------------|----|------------------------|
| 8 | Starting Python for the First Time | 20 | Working with Variables |
| 10 | Your First Code | 22 | User input |
| 12 | Saving and Executing Your Code | 24 | Creating Functions |
| 14 | Executing Code from the Command Line | 26 | Conditions and Loops |
| 16 | Numbers and Expressions | 28 | Python Modules |
| 18 | Using Comments | 30 | Python Errors |
| | | 32 | Python Graphics |

34 Mastering Linux

- | | | | |
|----|------------------------------------|----|---------------------------------|
| 36 | Using the Filesystem | 50 | Getting to Know Users |
| 38 | Listing and Moving Files | 52 | Ownership and Permissions |
| 40 | Creating and Deleting Files | 54 | Useful System and Disk Commands |
| 42 | Create and Remove Directories | 56 | Managing Programs and Processes |
| 44 | Copying, Moving and Renaming Files | 58 | Input, Output and Pipes |
| 46 | Using the Man Pages | 60 | Command Line Quick Reference |
| 48 | Editing Text Files | | |

62 Pi Projects

- | | | | |
|----|--------------------------------|----|-------------------------------------|
| 64 | Creating a Loading Screen | 76 | Pi Projects: Desktop Pi |
| 66 | Tracking the ISS with Python | 78 | Pi Projects: BBS Client |
| 70 | Text Animations | 80 | Pi Projects: Weather Station |
| 72 | Retro Coding | 82 | Stream Digital TV with Raspberry Pi |
| 74 | Using Text Files for Animation | 84 | Common Raspberry Pi Problems |



FREE CODE DOWNLOAD!

Contents



50 Complete Programs Over 20,000 lines of code!

Visit: www.pclpublications.com/exclusives

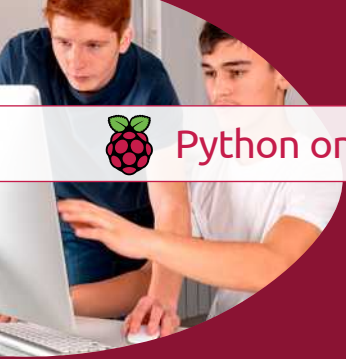
Please note: Sign up is required to download



“Thanks to the Raspberry Pi’s unique setup, operating system and amazing community of imagineers, this little computer has become one of the most important pieces of educational technology in the world. We’ve got everything from setting up the Pi to learning how to code on it using Python, and from learning Linux to projects that’ll take you and your Pi to the next level!”



Python on the Pi



Python On The Raspberry Pi

Being able to code is part of making not just the Raspberry Pi, but all the connected devices that can access the Internet, do what you want them to. The Raspberry Pi is a perfect coding base that comes pre-installed with the latest version of one of the world's most popular programming languages, Python.

Python is a powerful yet easy to understand programming language that enables you to do anything, from displaying simple messages on the screen to producing action-packed arcade games. In this section of the book, you will learn how to get Python working and how to code your first Python program.

Learning the tricks, hacks and fixes of Python will enable you to create something brilliant that can be shared with everyone who owns a Raspberry Pi.



Starting Python for the First Time

If you're using the Raspberry Pi, together with its Raspberry Pi OS, then you will need to manually install the Python IDLE. This is due to the Pi team removing the core Python IDLE in favour of replacing it with their own coding text editor.

STARTING PYTHON

For those using the Pi 4 and new Raspberry Pi OS, drop into a Terminal and enter: `sudo apt-get install idle3`. Older versions of Raspberry Pi OS already have the official Python IDLE pre-installed.

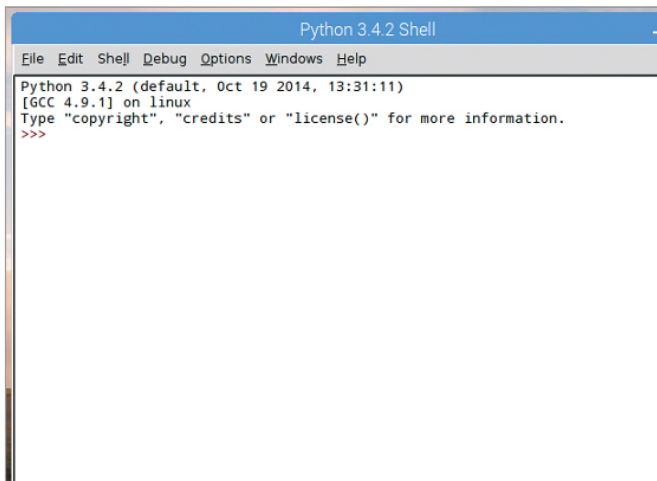
STEP 1

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



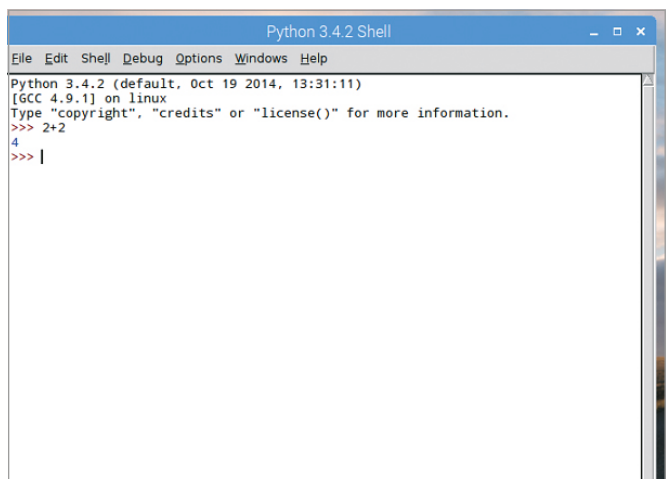
STEP 2

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



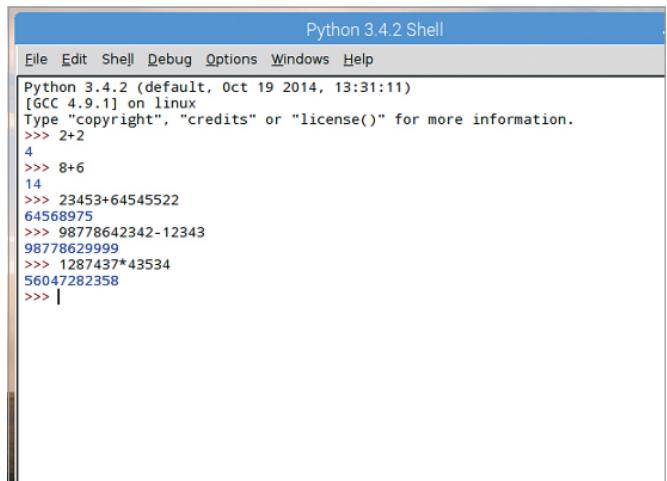
STEP 3

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



STEP 4

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





STEP 5

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

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

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

```
Python 3.4.2 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.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

STEP 6

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

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

STEP 7

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

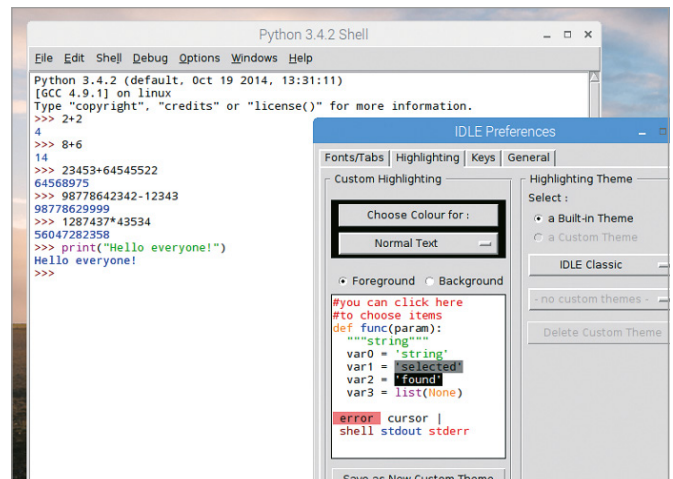
- | | |
|----------------------------|----------------------------|
| Black – Data and Variables | Blue – User Functions |
| Green – Strings | Dark Red – Comments |
| Purple – Functions | Light Red – Error Messages |
| Orange – Commands | |

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:

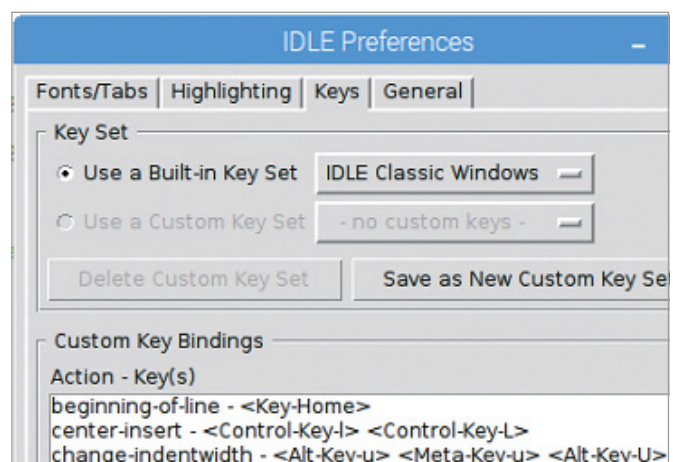
STEP 8

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.



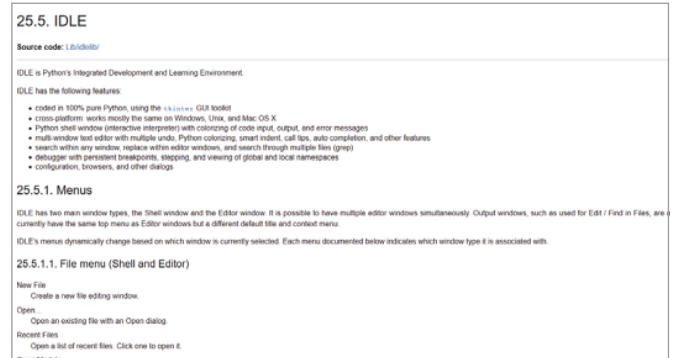
STEP 9

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





Your First Code

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

PLAYING WITH PYTHON

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

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

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

```
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
>>> |
```

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")
```

```
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
>>> |
```

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)
```

```
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
>>> |
```

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

```
a=2
```

```
b=2
```

```
Python 3.4.2 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
>>> |
```

**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.

```
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
>>> |
```

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 ourselves:

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

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

```
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
>>> print(name, surname)
David Hayward
>>> print(name+" "+surname)
David Hayward
>>> |
```




Saving and Executing Your Code

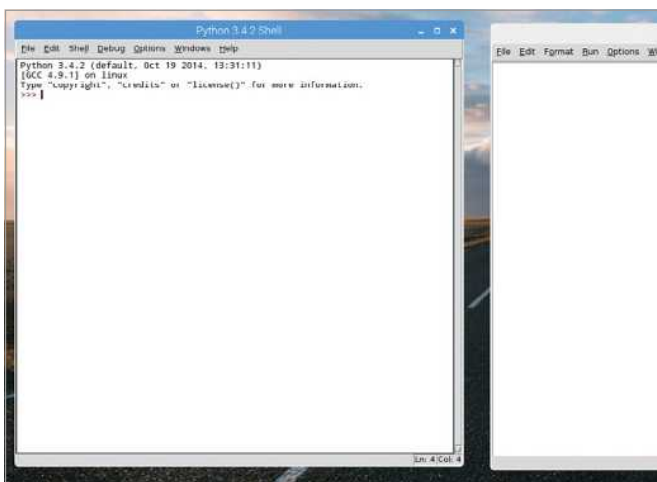
While working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

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

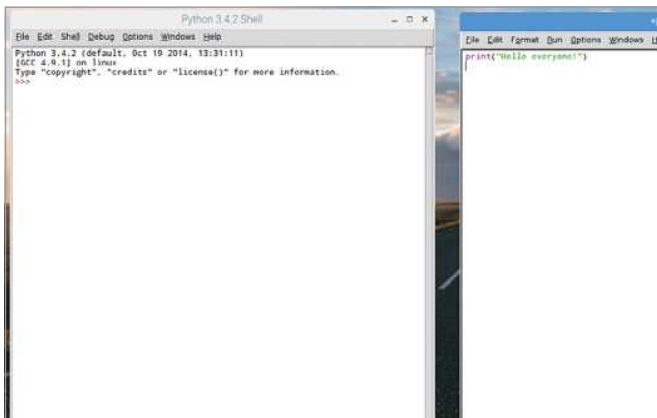
First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.



STEP 2

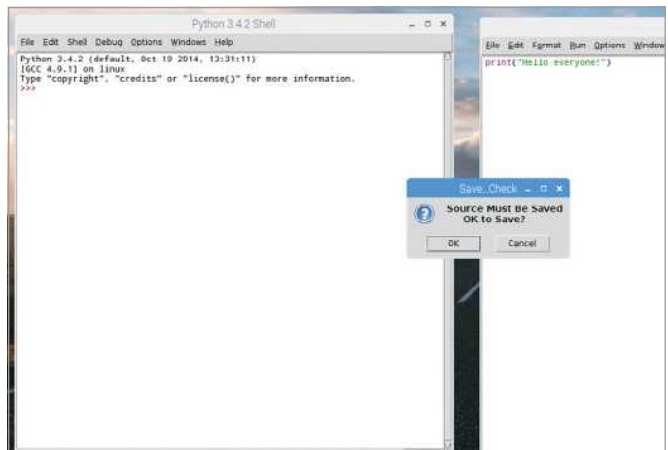
The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

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



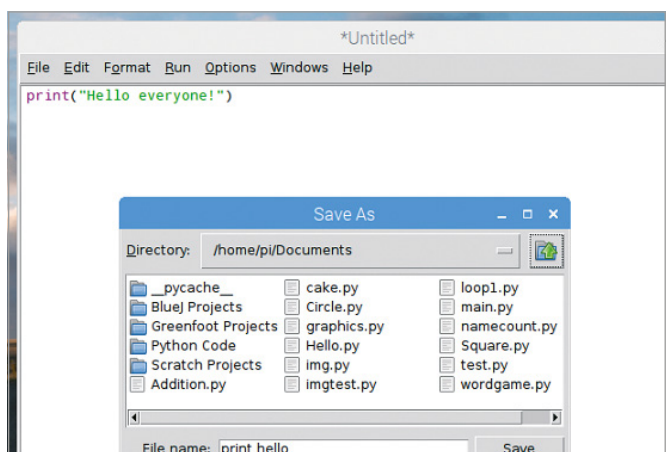
STEP 3

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

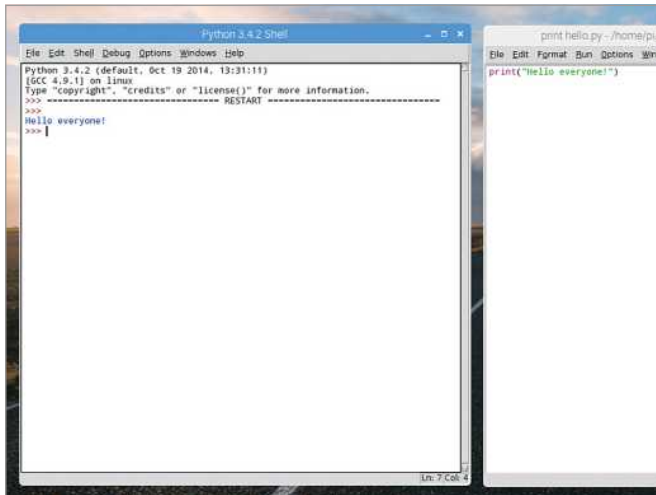
Click on the OK button in the Save box and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python or you can just dump it wherever you like. Remember to keep a tidy drive though, to help you out in the future.





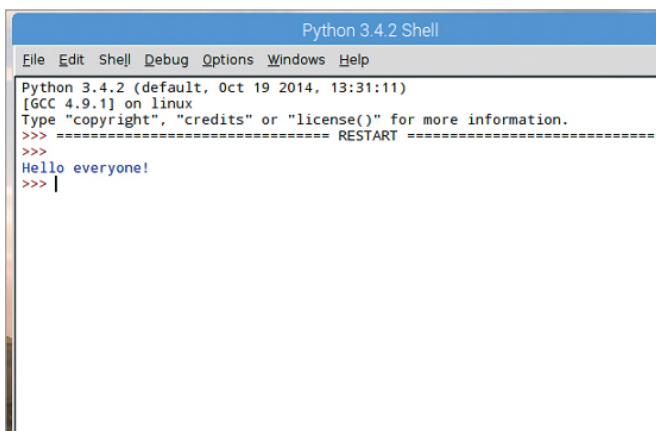
STEP 5

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!'.



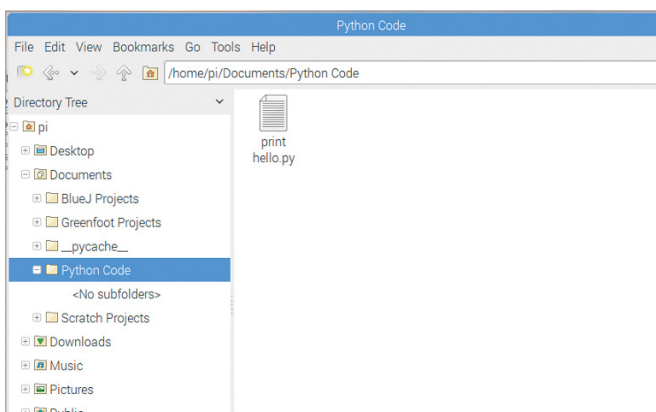
STEP 6

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.

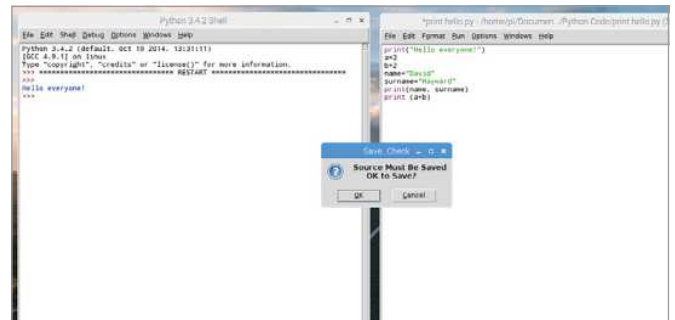


STEP 8

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

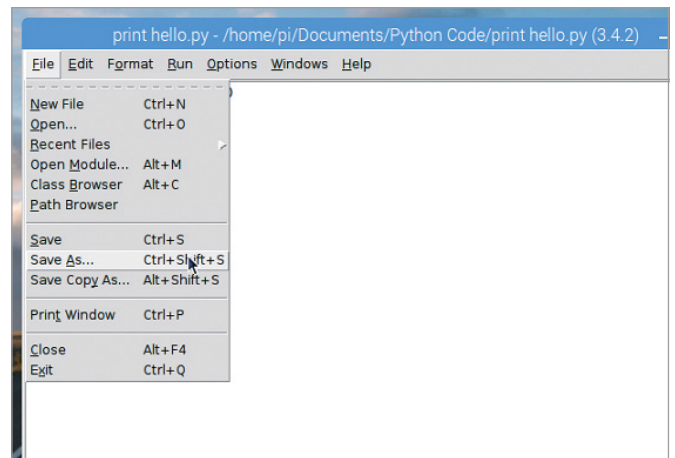
```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print(a+b)
```

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.

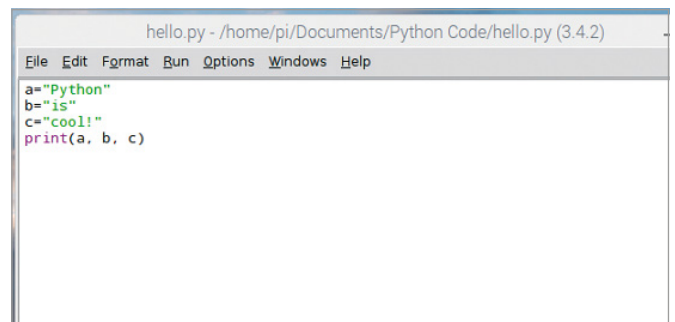


STEP 10

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

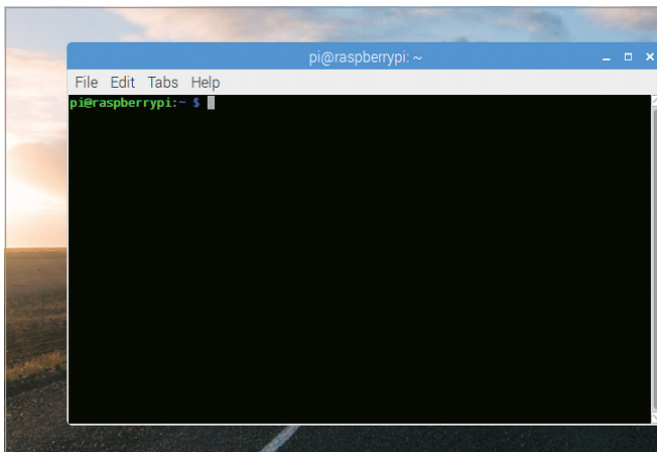
Although we're working from the GUI IDLE throughout this book, it's worth taking a look at Python's command line handling. We already know there's a command line version of Python but it's also used to execute code.

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.

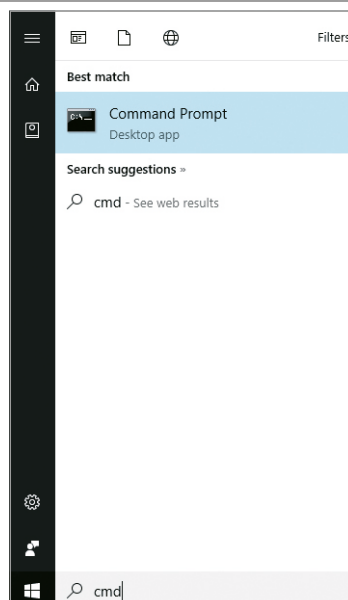
STEP 1

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.



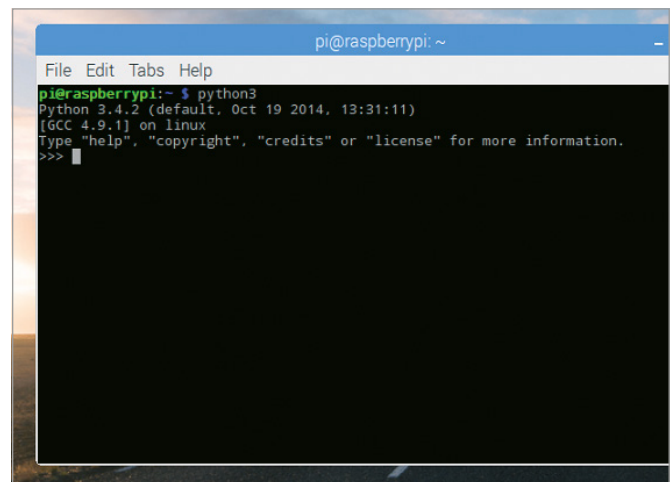
STEP 2

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 (`>>>`).



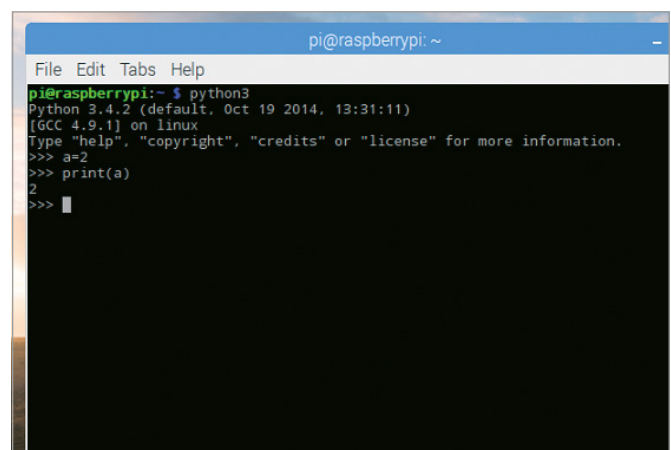
STEP 4

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.

```
pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help
pi@raspberrypi:~$ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~$ cd Documents/
pi@raspberrypi:~/Documents$ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code$ ls
hello.py print hello.py
pi@raspberrypi:~/Documents/Python Code$
```

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)
```

```
pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help
pi@raspberrypi:~$ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~$ cd Documents/
pi@raspberrypi:~/Documents$ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code$ ls
hello.py print hello.py
pi@raspberrypi:~/Documents/Python Code$ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code$
```

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`

```
pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help
pi@raspberrypi:~$ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~$ cd Documents/
pi@raspberrypi:~/Documents$ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code$ ls
hello.py print hello.py
pi@raspberrypi:~/Documents/Python Code$ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code$ python hello.py
('Python', 'is', 'cool!')
```

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.

```
C:\Users\david\Documents\Python\hello.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

hello.py
1 a="Python"
2 b="is"
3 c="cool!"
4 print(a, b, c)
5
```

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)
```

```
C:\Users\david\Documents\Python\hello.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

hello.py
1 a="Python"
2 b="is"
3 c="cool!"
4 print(a, b, c)
5 name=input("What is your name? ")
6 print("Hello,", name)
7
```

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.

```
pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help
pi@raspberrypi:~/Documents/Python Code$ python3 hello.py
Python is cool!
What is your name? David
Hello, David
pi@raspberrypi:~/Documents/Python Code$
```




Numbers and Expressions

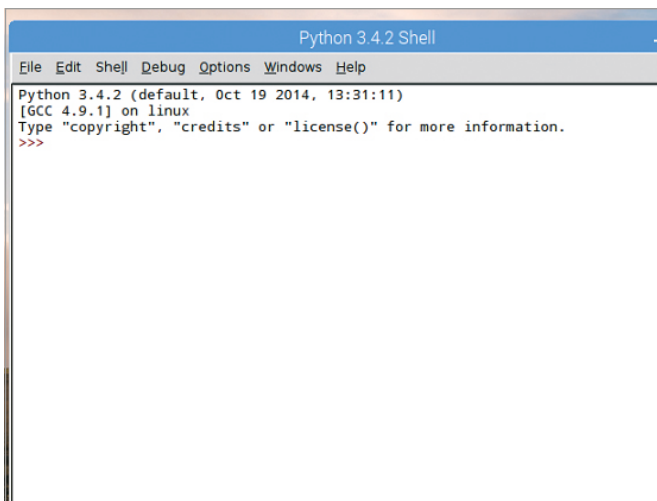
We've seen some basic mathematical expressions with Python, simple addition and the like. Let's expand on that now and see just how powerful Python is as a calculator. You can work within the IDLE Shell or in the Editor, whichever you like.

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).



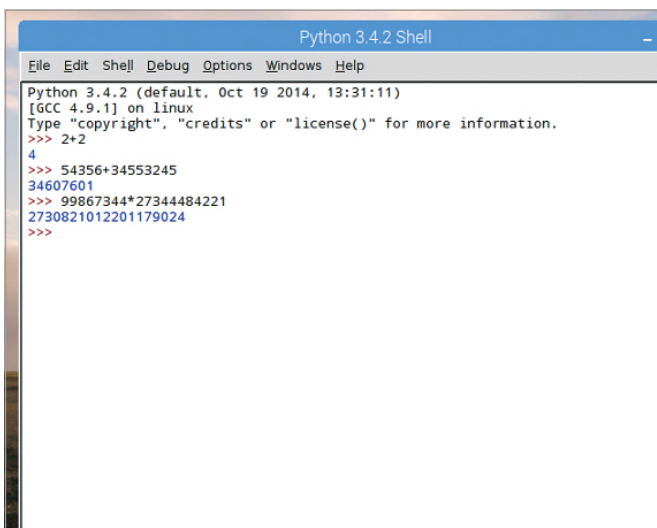
```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

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.

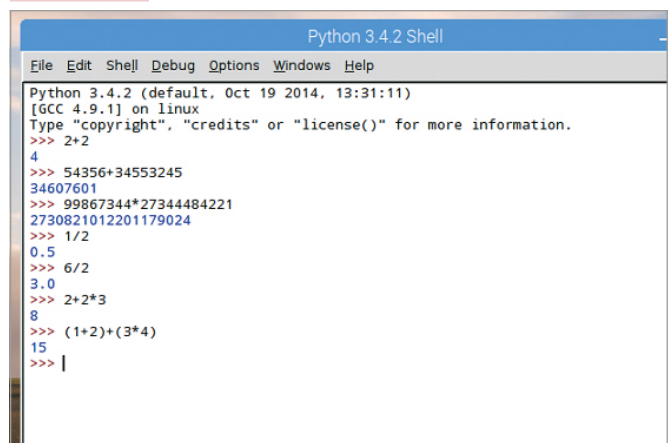


```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>>
```

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)
```

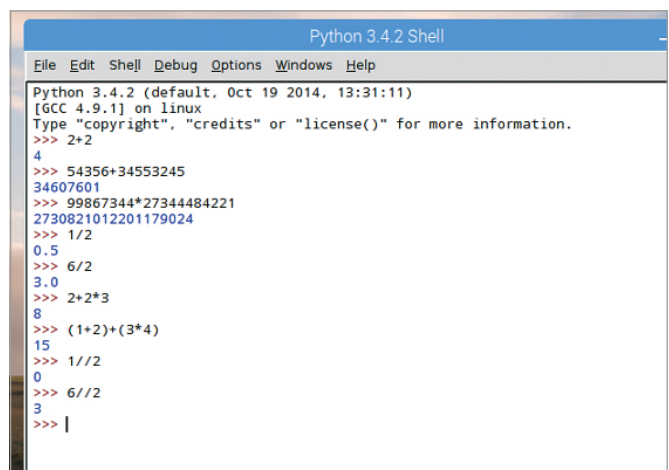


```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> |
```

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.
```



```
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.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> |
```

**STEP 5**

You can also use an operation to see the remainder left over from division. For example: `10/3`

Will display `3.3333333333333333`, which is of course 3.3-recurring.

If you now enter: `10%3`

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

```
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.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333333
>>> 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.

```
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333333
>>> 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)`

```
>>> 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
3.3333333333333333
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> |
```

STEP 8

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

`format(3, 'b')`

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

```
Python 3.4.2 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.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333333
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3, 'b')
'11'
>>>
```

STEP 9

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

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

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

STEP 10

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

```
Python 3.4.2 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 =====
>>>
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>> |
```




Using Comments

When writing your code, the flow, what each variable does, how the overall program will operate and so on is all inside your head. Another programmer could follow the code line by line but 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

While 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.

```
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 =====
>>> The value of A is, 10
>>> ===== RESTART =====
>>> The value of A is, 10
>>> ===== RESTART =====
>>> The value of A is, 10
You've just lost a life!
You now have 9 lives left!
>>>
```

STEP 6

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

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

```
*Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)*
File Edit Format Run Options Windows Help
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current 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.
```

```
*Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)*
File Edit Format Run Options Windows Help
# This is the best game ever, and has been developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite being very smelly, the code at least
# works really well.
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 8

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

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

```
Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform player, and display current 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)
```

```
*Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 10

You also use three single quotes to comment out a Block Comment or multi-line section of comments. 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.
'''
```

```
*Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)*
File Edit Format Run Options Windows Help
'''
This is the best game ever, and has been developed by a crack squad of Python experts
who haven't slept or washed in weeks. Despite being very smelly, the code at least
works really well.
'''
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```



Working with Variables

Variables are where all the action takes place in your code. A Variable can be anything, from the number of lives in a game, that changes as you progress through the game, to a person's name, age and address. Variables can be static or alter depending on what you want the code to do.

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 are integers, as we know.

```
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
>>> character=name + ": " + title
>>> print(character)
David Hayward: Descended from Vikings
>>> age=44
>>> type(age)
```


**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 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
>>> 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()**.

```
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.
>>> len()
```

STEP 2

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

```
Python 3.4.2 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.
>>> len("antidisestablishmentarianism")
28
>>> |
```

STEP 3

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

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

Press F5 and save the code to execute it.

```
namecount.py - Runwip2.Docus
File Edit Format Run 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.
>>> len("antidisestablishmentarianism")
28
>>> ===== RESTART =====
Enter your surname: Hayward
Your name has 7 letters in it.
>>> |
```

STEP 4

Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to 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.

```
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.
>>> len("antidisestablishmentarianism")
28
>>> ===== RESTART =====
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>> |
```



STEP 5

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

import math

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

```
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.
>>> len("antidisestablishmentarianism")
28
>>> ===== RESTART =====
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

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).

```
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.
>>> len("antidisestablishmentarianism")
28
>>> ===== RESTART =====
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>> math.sqrt(16)
4.0
>>> |
```

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.

```
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 =====
>>>
Hello
>>> Hello()
Hello
>>> |

Hello.py - /f
File Edit Format Run Options Help
def Hello():
    print("Hello")
Hello()
```

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.

```
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 =====
>>>
Hello David
>>> name="Bob"
>>> Hello(name)
Hello Bob
>>> |

Hello.py - /f
File Edit Format Run Options Help
def Hello(name):
    print("Hello", name)
Hello("David")
```

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.

```
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 =====
>>>
Hello David
>>> |

test.py - /home/p
File Edit Format Run Options Help
from Hello import Hello
Hello("David")
```

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
```

```
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 =====
>>>
Hello David
>>> name="Bob"
>>> Hello(name)
Hello Bob
>>> add(10,12.2)
10.2
>>> add(219.22,28)
247.22
>>> |

Addition.py - /home/p/Documents/Addition
File Edit Format Run Options Help
def add(a, b):
    result = a + b
    return result
```



Conditions and Loops

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

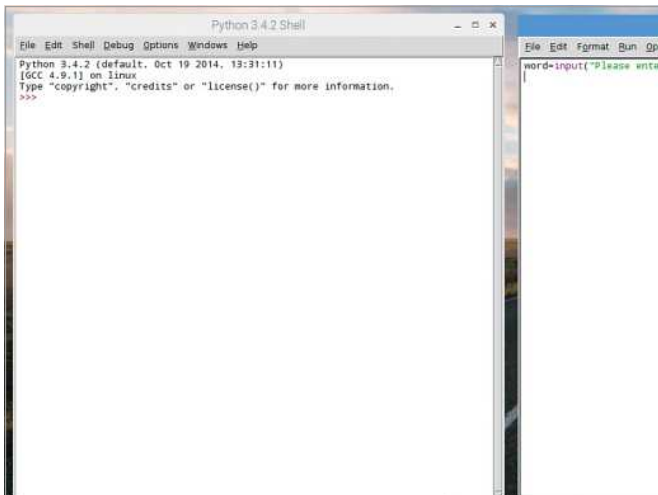
TRUE CONDITIONS

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

STEP 1

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

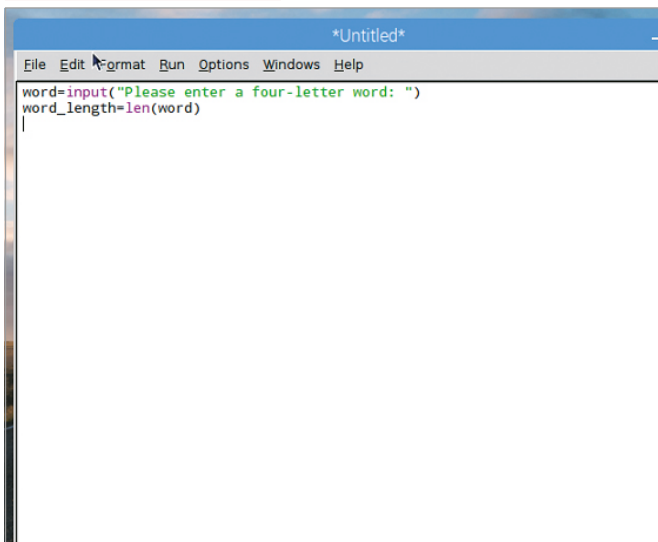
```
word=input("Please enter a four-letter word: ")
```



STEP 2

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

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

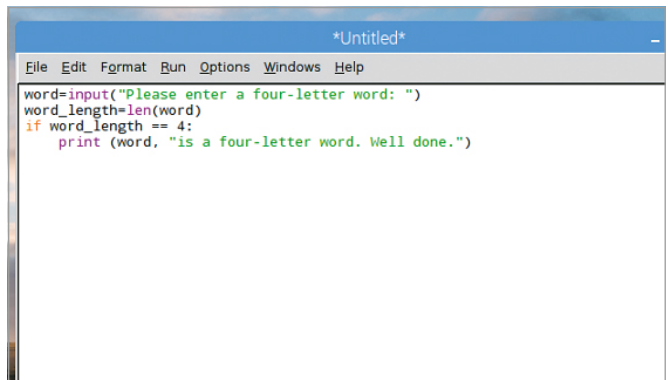


STEP 3

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

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

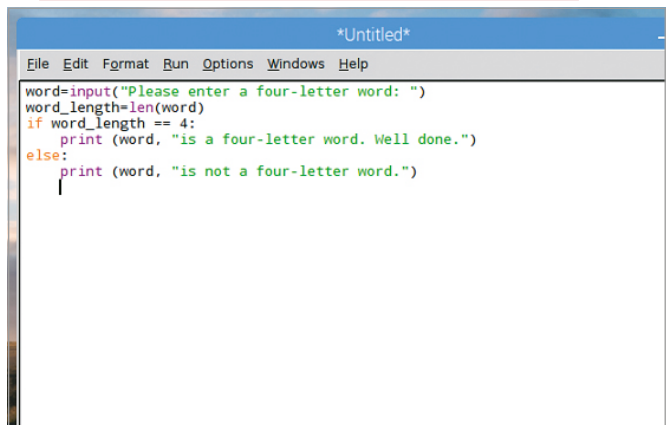
The double equal sign (==) means check if something is equal to something else.



STEP 4

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

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





STEP 5

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

```
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 =====
>>> Please enter a four-letter word: word
word is a four-letter word. Well done.
>>> Please enter a four-letter word: Frost
Frost is 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.")
```

```
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 =====
>>> Please enter a four-letter word: word
word is a four-letter word. Well done.
>>> Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> Please enter a four-letter word: Egg
Egg is a three-letter word. Try again.
>>>
```

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
```

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

STEP 3

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

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

```
Python 3.4.2 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 =====
>>>
1
2
3
4
5
6
7
8
9
>>>
Cat
Dog
Unicorn
>>>
```

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.

```
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 =====
>>>
1
2
3
4
5
6
7
8
9
>>>
```

STEP 4

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

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

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

```
Python 3.4.2 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 =====
>>>
1
2
3
4
5
6
7
8
9
>>>
```



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 -----
14
21
3
17
22
4
8
5
10
13
>>> ----- RESTART: C:/Users/david/Documents/Python/Rnd Number.py -----
26
11
17
65
37
52
37
38
89
54
42
48
96
```

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
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
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
File Edit Format Run Options Window Help

import math as m

print(m.pi)
```




Python Errors

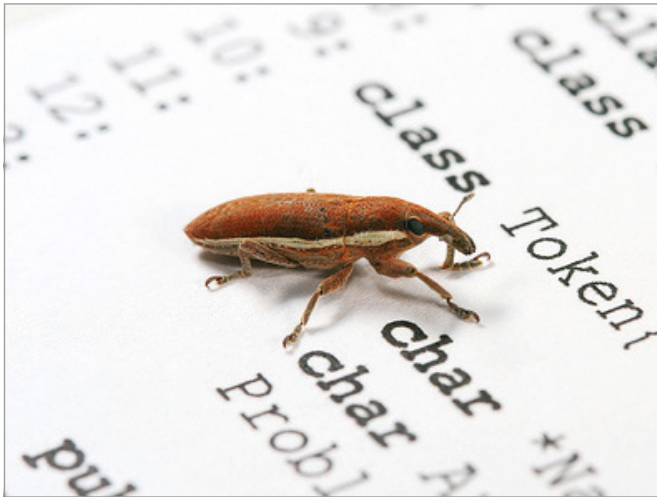
It goes without saying that you'll eventually come across an error in your code, where Python declares it's not able to continue due to something being missed out, wrong or simply unknown. Being able to identify these errors makes for a good programmer.

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.

```
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
>>> apples=10
>>> pirnt(apples)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    pirnt(apples)
NameError: name 'pirnt' is not defined
>>> |
```

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.

```
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 =====
>>>
Traceback (most recent call last):
  File "/home/pi/Documents/graphics.py", line 83, in <module>
    fgpygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
NameError: name 'fgpygame' is not defined
>>>
```

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"**.

```
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 world!"
SyntaxError: invalid syntax
>>>
```



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:
21            return "NO"
22
23 if len(stack) == 0:
24     return "YES"
```

STEP 6

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

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

```
8 try:
9     some_code()
10 except:
11     clean_up()
```

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

A few other general comments on your code:

- In line 200, you have this construction:

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

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

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.

4letterwrld.py - /home/pi/Documents/Python Code/4letterwrld.py (3.4.2)

```
File Edit Format Run Options Windows Help

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

SyntaxError - expected an indented block

OK

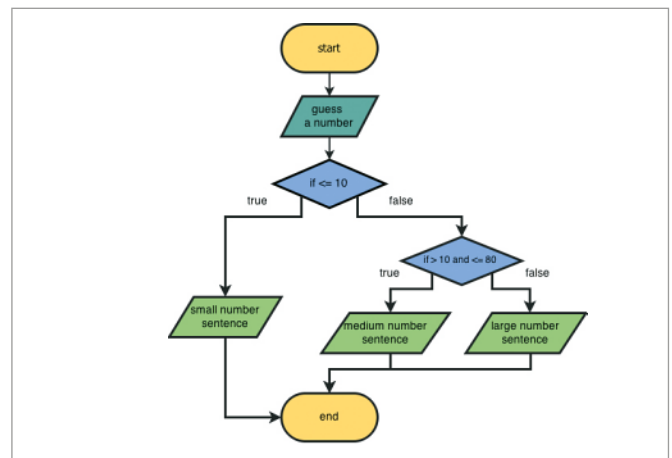
STEP 8

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



STEP 9

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



STEP 10

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





Python Graphics

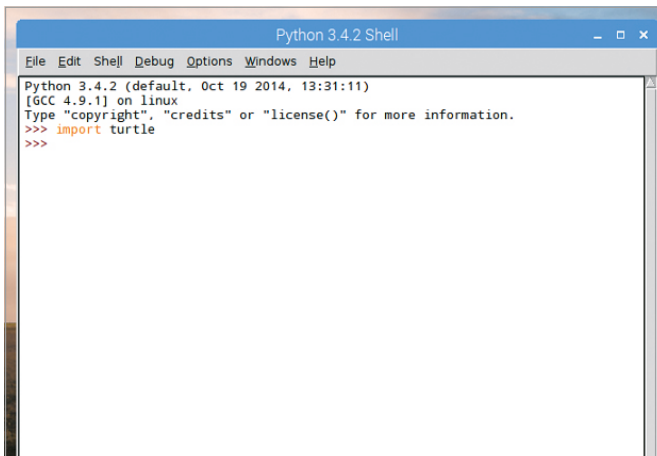
While dealing with text on the screen, either as a game or in a program, is great, there will come a time when a bit of graphical representation wouldn't go amiss. Python 3 has numerous ways in which to include graphics and they're surprisingly powerful too.

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`.

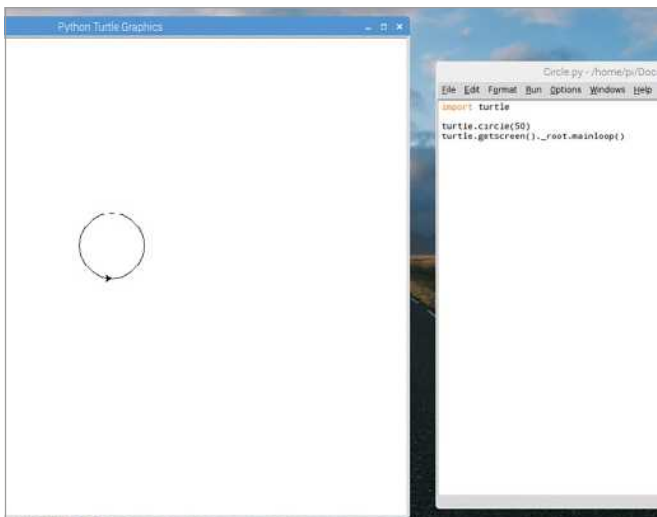


STEP 2

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

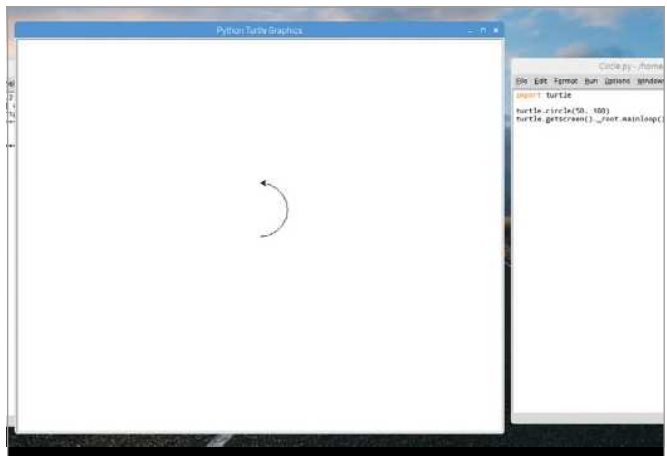
```
import turtle
turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

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.

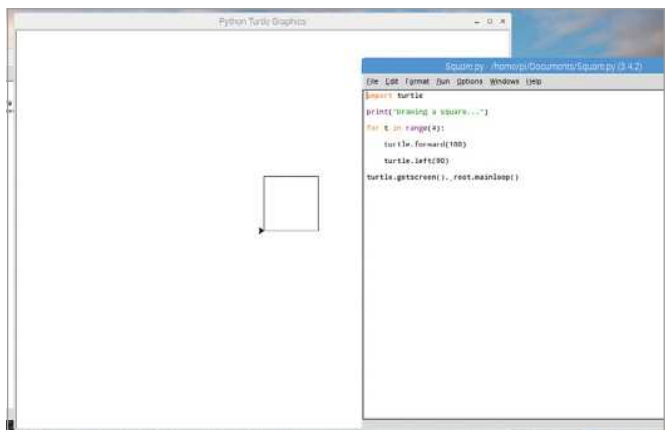


STEP 4

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")
```

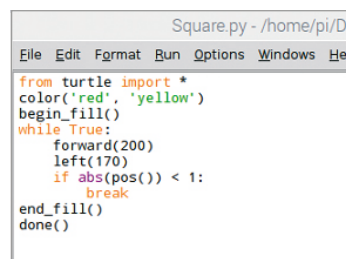
You can also use the command `turtle.begin_fill()`, and `turtle.end_fill()` to fill in the square with the chosen colours; red outline, yellow fill in this case.

**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**

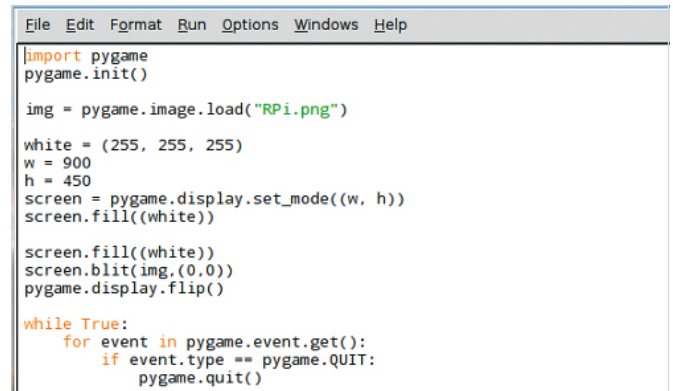
Another way in which you can display graphics is by using the Pygame module. There are numerous ways

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.blit(img, (0,0))
pygame.display.flip()
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

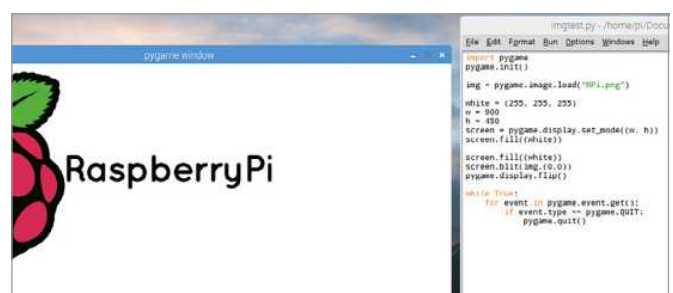
**STEP 9**

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.

```
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.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.





Mastering Linux On Pi

The Raspberry Pi OS is Linux based, which is an incredibly powerful operating system that drives many supercomputers, web servers and even top-end military hardware. Get to grips with Linux, and you will not just master the Pi but also the power behind the Internet.

In this section, you will discover how the OS works, how the filesystem is built and how you can list, move, create and delete files and folders. To truly be able to master the Raspberry Pi, you will need to be familiar with the OS and its inner-workings.

The more adept you become at Linux, the better your projects will be, and the more power you'll have over them and how they interact with the digital world to which they're connected.



Using the Filesystem

To master Linux, it's important to understand how the filesystem works. What's more, it's also important to become familiar with the Terminal, or shell. This command line environment may appear daunting at first, but with practise, it soon becomes easy to use.

GETTING AROUND

To drop into the Terminal, click on the fourth icon from the left along the top of the Raspberry Pi desktop, the one with a right-facing arrow and an underscore. This is the shell, or Terminal.

STEP 1

First, you're going to look at directories and the directory path. A directory is the same thing as a folder, however in Linux it's always called a directory. These are placed inside each other using a "/" character. So when you see /home/pi it means the pi directory is inside the home directory. Enter: `clear` and press return to clean the screen. Now enter: `pwd`. This stands for Print Working Directory and displays /home/pi.

```
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $
```

STEP 2

When you log in to your Raspberry Pi, you don't start at the base of the hard drive, known as the 'root' (also known as the topmost directory). Instead you begin inside your user directory, which is named 'pi' by default and is itself in a directory called 'home'. Directories are indicated by the '/' symbol. So, "/home/pi" tells you that in the root is a directory called home, and the next "/" says that inside "home" is a directory called "pi". That's where you start.

```
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $
```

STEP 3

Enter: `ls` to view the contents of the current directory. You should see Desktop, Documents, and Downloads and Scratch in Blue. You may also see other items depending on how much you have used your Raspberry Pi. The colour code is worth knowing: directories are blue while most files are white. As you go on you'll see other colours: executable files (programs) are bright green, archived files are red and so on. Blue and white are the two you need to know to get started.

```
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $ ls
Desktop Documents Downloads exit indiecity python_games Scratch
```

STEP 4

Now you're going to move from the pi directory into the Documents directory. Enter: `cd Documents`. Note the capital "D". Linux is case sensitive, which means you have to enter the exact name including correct capitalisation. The `cd` command stands for change directory. Now enter: `pwd` again to view the directory path. It will display /home/pi/ Documents. Enter: `ls` to view the files inside the Documents directory.

```
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $ ls
Desktop Documents Downloads exit indiecity python_games Scratch
pi@raspberrypi ~ $ cd Documents
pi@raspberrypi ~/Documents $ pwd
/home/pi/Documents
pi@raspberrypi ~/Documents $ ls
amber archive.tar Crypto101.pdf dog_jump euro fizzbang_backup.py fizzbang.py
pi@raspberrypi ~/Documents $
```

**STEP 5**

How do you get back up to the pi directory? By using a command "cd ..". In Linux two dots means the directory above, also known as the parent directory. Incidentally, a single dot "." is used for the same directory. You never use "cd ." to switch to the same directory but it's worth knowing because some commands need you to specify the current directory.

```
pi@raspberrypi ~/Documents $ pwd
/home/pi/Documents
pi@raspberrypi ~/Documents $ cd ..
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $
```

STEP 6

The "ls" and "cd" commands can also be used with more complex paths. Enter: **ls Documents/Pictures** to view the contents of a Pictures directory inside your Documents directory. You can switch to this directory using **cd Documents/Pictures**; use **cd ../../** to move back up two parent directories.

```
pi@raspberrypi ~ $ ls Documents/Pictures
LEGO LucyHattersley.jpg raspberry_pi_2_photographs
pi@raspberrypi ~ $ cd Documents/Pictures
pi@raspberrypi ~/Documents/Pictures $ pwd
/home/pi/Documents/Pictures
pi@raspberrypi ~/Documents/Pictures $ cd ../../
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $
```

ABSOLUTE VS RELATIVE PATHS

It is important to know the difference between the working directory, root directory and home. There are also two types of path: Absolute and Relative. These are easier to understand than they sound. Let's take a look...

STEP 1

By default, commands like "ls" use the working directory. This is the current directory that you're looking at and is set to your home directory by default (/users/pi). Using "pwd" (Print Working Directory) lets you know what the working directory is, and using "cd" changes the working directory.

```
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $
```

STEP 3

The second command ("ls /Documents/Pictures") attempts to list the content of Pictures in a directory called Documents inside the root directory (because the path started with '/', which is root). There is typically no Documents directory in root, so you will get a "No such file or directory" error. Starting a path with '/' is known as an "absolute path", while starting without the '/' is known as a "relative path" because it is relative to your working directory.

```
pi@raspberrypi ~ $ ls /
bin boot dev etc home lib lost+found media mnt opt pro
pi@raspberrypi ~ $ ls /Documents/Pictures
ls: cannot access /Documents/Pictures: No such file or directory
pi@raspberrypi ~ $
```

STEP 2

The root directory is always '/'. Entering: **ls /** lists the contents of root, and entering: **cd /** switches to the root directory. This is important because there is a difference between "ls Documents/Pictures" and "ls /Documents/Pictures". The first command lists the contents of the Pictures directory in Documents inside the working directory (which, if you are in the home directory, will work).

```
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ ls Documents/Pictures
BDM-Web-logo-dark1.jpg David Hayward.jpg RPi.png
pi@raspberrypi:~ $
```

STEP 4

There is also an absolute path shortcut to your user directory, and that is the tilde "~" character. Entering: **ls ~** always lists the contents of your home directory, while "cd ~" moves straight to your home directory, no matter what your working directory is. You can also use this shortcut wherever you are: enter: **ls ~/Documents/Pictures** to display the contents of the Pictures.

```
pi@raspberrypi:~ $ cd ~
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ ls ~/Documents/Pictures
BDM-Web-logo-dark1.jpg David Hayward.jpg RPi.png
pi@raspberrypi:~ $
```



Listing and Moving Files

Admittedly, using the desktop GUI to list and move files is much easier than using the Terminal and keyboard. However, it's an important skill that you will appreciate as you advance with the Raspberry Pi and Linux.

LOOKING AT FILES

Operating systems are built on files and folders, or directories if you prefer. While you're used to viewing your own files, most operating systems keep other files out of sight. In Raspberry Pi OS, you have access to every file in the system.

STEP 1

We've already looked at "ls", which lists the files in the working directory, but you are more likely to use a command like "ls -l". The bit after the command (the '-l' option) is known as the argument. This is an option that modifies the behaviour of the command.

```
pi@raspberrypi ~ $ ls -l_
```

STEP 2

The "-l" argument lists files and directories in long format. Each file and directory is now on a single line, and before each file is a lot of text. First you'll see lots of letters and dashes, like 'drwxr-xr-x'. Don't worry about these for now; they are known as 'permissions' and we'll come to those later.

```
pi@raspberrypi ~ $ ls -l
total 24
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Apr 21 14:50 Documents
drwx----- 2 pi pi 4096 Apr 21 15:23 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
-rw-r--r-- 1 pi pi 0 May 11 20:56 names.txt
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Apr 17 12:53 Scratch
pi@raspberrypi ~ $
```

STEP 3

After the permission letters come a single number. This is the number of files in the item. If it's a file then it'll be 1, but if it's a directory it'll be at least 2. This is because each directory contains two hidden files; one with a single dot (.) and one with two dots (.). Directories containing files or other directories will have a higher number.

```
pi@raspberrypi ~ $ ls -l
total 24
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Apr 21 14:50 Documents
drwx----- 2 pi pi 4096 Apr 21 15:23 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
-rw-r--r-- 1 pi pi 0 May 11 20:56 names.txt
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Apr 17 12:53 Scratch
pi@raspberrypi ~ $
```

STEP 4

Next you'll see the word "pi" listed twice on each line. This refers to the user rather than the name of your computer (your default username is "pi"). The first is the owner of the file, and the second is the group. Typically these will both be the same and you'll see either 'pi' or 'root'. You can enter: **ls -l /** to view the files and directories in the root directory that belong to the root account.

```
pi@raspberrypi ~ $ ls -l
total 28
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Apr 21 14:50 Documents
drwx----- 2 pi pi 4096 Apr 21 15:23 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
-rw-r--r-- 1 pi pi 0 May 11 20:56 names.txt
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Apr 17 12:53 Scratch
drwxr-xr-x 3 pi pi 4096 May 11 21:15 test
pi@raspberrypi ~ $ ls -l /
total 74
drwxr-xr-x 2 root root 4096 Jan 1 1970 bin
drwxr-xr-x 3 root root 2048 Jan 1 1970 boot
drwxr-xr-x 12 root root 3280 May 11 09:03 dev
drwxr-xr-x 109 root root 4096 May 11 09:03 etc
drwxr-xr-x 3 root root 4096 Jan 1 1970 home
drwxr-xr-x 12 root root 4096 Jan 1 1970 lib
drwx----- 2 root root 16384 Feb 15 11:21 lost+found
drwxr-xr-x 3 root root 4096 May 11 07:42 media
drwxr-xr-x 2 root root 4096 Jan 1 00:02 mnt
drwxr-xr-x 6 root root 4096 Jan 1 1970 opt
dr-xr-xr-x 86 root root 0 Jan 1 1970 proc
drwx----- 9 root root 4096 May 11 07:36 root
drwxr-xr-x 10 root root 460 May 11 09:03 run
drwxr-xr-x 2 root root 4096 Jan 1 1970/sbin
```




STEP 5

The next number relates to the size of the file, in bytes.

In Linux each text file is made up of letters and each letter takes up a byte, so our names.txt file has 37 bytes and 37 characters in the document. Files and directories can be extremely large and hard to determine, so use "ls -lh". The "h" argument humanises the number, making it easier to read.

```
pi@raspberrypi ~ $ ls -l
total 32
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
drwxr-xr-x 2 pi pi 4.0K Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4.0K Apr 21 14:50 Documents
drwx----- 2 pi pi 4.0K Apr 21 15:23 Downloads
drwxr-xr-x 3 pi pi 4.0K Apr 17 18:48 indiecity
-rw-r--r-- 1 pi pi 37 May 11 21:27 names.txt
drwxr-xr-x 2 pi pi 4.0K Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4.0K Apr 17 12:53 Scratch
drwxr-xr-x 3 pi pi 4.0K May 11 21:15 test
pi@raspberrypi ~ $ ls -lh
total 32K
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
drwxr-xr-x 2 pi pi 4.0K Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4.0K Apr 21 14:50 Documents
drwx----- 2 pi pi 4.0K Apr 21 15:23 Downloads
drwxr-xr-x 3 pi pi 4.0K Apr 17 18:48 indiecity
-rw-r--r-- 1 pi pi 37 May 11 21:27 names.txt
```

STEP 6

Finally, you should be aware that there are many hidden files in Linux. These are listed using the "-a" argument. Hidden files and directories begin with a dot (.), so you should never start a file or directory with a dot, unless you want to hide it. Typically, you can combine all three arguments together into the command "ls -lah".

```
pi@raspberrypi ~ $ ls -lah
total 520K
drwxr-xr-x 33 pi pi 4.0K May 11 21:14 .
drwxr-xr-x 3 root root 4.0K Jan 1 1970 ..
drwx----- 2 pi pi 4.0K Apr 20 14:31 .aptitude
-rw-r--r-- 1 pi pi 0 May 11 20:56 articles.txt
-rw-r--r-- 1 pi pi 8.7K May 11 09:03 bash_history
-rw-r--r-- 1 pi pi 220 Feb 15 14:05 bash_logout
-rw-r--r-- 1 pi pi 3.2K Feb 15 14:05 bashrc
drwxr-xr-x 10 pi pi 4.0K Apr 21 17:08 cache
drwxr-xr-x 20 pi pi 4.0K Apr 21 13:33 config
drwx----- 3 pi pi 4.0K Feb 16 14:16 dbus
drwxr-xr-x 2 pi pi 4.0K Apr 21 17:55 Desktop
-rw-r--r-- 1 pi pi 35 Apr 17 12:17 darc
drwxr-xr-x 5 pi pi 4.0K Apr 21 14:50 Documents
drwx----- 2 pi pi 4.0K Apr 21 15:23 Downloads
drwxr-xr-x 2 pi pi 4.0K Apr 20 13:45 dreamchess
drwxr-xr-x 2 pi pi 4.0K Apr 21 18:15 fontconfig
drwxr-xr-x 2 pi pi 4.0K Apr 17 18:56 freeciv
```

SOME COMMON DIRECTORIES

Now that you know how to view the contents of your hard drive you'll start to notice a lot of directories with names like bin,/sbin, var and dev. These are the files and directories that you are kept away from on a Mac, and won't encounter on a Windows PC.

STEP 1

Enter: `ls -lah /` to view all of the files and directories, including the hidden items, in the root

directory of your hard drive. Here you will see all the items that make up your Raspberry Pi OS (which is a version of Linux). It's worth taking the time to know some of them.

```
pi@raspberrypi ~ $ ls -lah /
total 82K
drwxr-xr-x 22 root root 4.0K May 11 21:23 .
drwxr-xr-x 22 root root 4.0K May 11 21:23 ..
drwxr-xr-x 2 root root 4.0K Jan 1 1970 bin
drwxr-xr-x 3 root root 2.0K Jan 1 1970 boot
drwxr-xr-x 12 root root 3.3K May 11 09:03 dev
drwxr-xr-x 109 root root 4.0K May 11 09:03 etc
drwxr-xr-x 3 root root 4.0K Jan 1 1970 home
drwxr-xr-x 12 root root 4.0K Jan 1 1970 lib
drwx----- 2 root root 16K Feb 15 11:21 lost+found
drwxr-xr-x 3 root root 4.0K May 11 07:42 media
drwxr-xr-x 2 root root 4.0K Jan 1 00:02 mnt
drwxr-xr-x 6 root root 4.0K Jan 1 1970 opt
dr-xr-xr-x 85 root root 0 Jan 1 1970 proc
drwx----- 9 root root 4.0K May 11 07:36 root
drwxr-xr-x 10 root root 460 May 11 09:03 run
drwxr-xr-x 2 root root 4.0K Jan 1 1970/sbin
drwxr-xr-x 2 root root 4.0K Jun 20 2012 selinux
drwxr-xr-x 2 root root 4.0K Feb 15 11:23 srv
dr-xr-xr-x 12 root root 0 May 11 21:00 sys
drwxr-xr-x 4 root root 4.0K May 11 21:39 tmp
```

STEP 2

Bin is a directory that stores binaries. This is the Linux way of saying programs or applications. Sbin is for system binaries, which are the programs that make up your system. Dev contains references to your devices: hard drive, keyboard, mouse and so on. Etc contains your system configuration files.

```
pi@raspberrypi ~ $ ls /bin
bash          bzfgrep      chgrp        dash          domainname   fgconsole
bunzip2       bzgrep      chmod        date          dumpkeys     fgrep
bcat          bzfp2       chown        dd            echo          findmnt
bcmp          bzfp2recover chut         df            ed            fuser
bdiff         bzless      con2fbmap    dir           egrep         fusemount
bzegrep       bzmore      cp           dmesg         false        grep
bzeke         cat         cpio         dnsdomainname fbset         gunzip
pi@raspberrypi ~ $
```

STEP 3

Entering: `ls /home` displays the contents of your home directory, which contains pi; the directory that you start in. So, entering: `ls /home/pi` is the same as just "ls" from the default home directory. This is where you are expected to place most of the documents you create. Don't confuse home with "usr"; the /usr directory is where you find program tools and libraries.

```
pi@raspberrypi ~ $ ls
articles.txt Desktop Documents Downloads indiecity names.txt python
pi@raspberrypi ~ $ ls /home/pi
articles.txt Desktop Documents Downloads indiecity names.txt python
pi@raspberrypi ~ $
```

STEP 4

Lib is a directory that contains libraries of code that are referred to by other programs (different programs share files in Lib). "Var" is short for various, which is mostly files used by the system, but you may need to work with items here. Finally there is a directory called "tmp", which is for temporary files; files placed here are on your system for the short term and can be deleted from the system.

```
pi@raspberrypi ~ $ ls /var
backups cache lib local lock log mail opt run spool swap tmp
pi@raspberrypi ~ $
```



Creating and Deleting Files

Being able to create and delete a file is an everyday computing skill. However, when using the Linux Terminal, there's an element of care required, chiefly because any deleted files aren't placed in the system recycle bin.

CREATING FILES

Once you learn to recognise the files and directories that make up Raspberry Pi OS OS, it's time to discover how to make your own. Knowing how to make, edit and delete files and directories is essential if you want to make your own projects.

STEP 1

We're going to create a file using a command called Touch. Touch is an interesting command that reaches out to a file, or directory, and updates it (this changes the system time as if you'd just opened the file). You can see Touch in access using "ls -l" and checking the time next to a directory (such as Scratch).

```
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Apr 17 12:53 Scratch
pi@raspberrypi ~ $ _
```

STEP 2

Now enter: `touch Scratch` and `ls -l` again and notice that the time has changed. It now matches the current time. You might be wondering what this has to do with creating files or directories. Touch has a second, more popular, use, which is to create files.

```
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Apr 17 12:53 Scratch
pi@raspberrypi ~ $ touch Scratch
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
pi@raspberrypi ~ $ _
```

STEP 3

If you try to touch a file that doesn't exist, you create a blank file with that name. Try it now. Type `touch testfile` and `ls -l` to view the files. You'll now have a new file in your home directory called "testfile". Notice that the size of the file is 0, because it has nothing in it.

```
pi@raspberrypi ~ $ touch testfile
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:10 testfile
pi@raspberrypi ~ $ _
```

STEP 4

A quick word about file names: remember that Linux is case sensitive, so if you now enter: `touch Testfile` (with a capital T), it doesn't update 'testfile'; instead, it creates a second file called 'Testfile'. Enter: `ls -l` to see both files. This is confusing, so most people stick with using lowercase letters at all times.

```
pi@raspberrypi ~ $ touch testfile
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:08 testfile
pi@raspberrypi ~ $ touch Testfile
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:08 testfile
-rw-r--r-- 1 pi pi 0 May 13 11:10 Testfile
pi@raspberrypi ~ $ _
```



STEP 5

Another important thing to know is never to use a space in your file names. If you try to enter: `touch test file`, you create a document called "test" and another called "file". Technically there are ways to create files containing a space but you should always use an underscore character ("_") instead of a space, such as "touch test_file".

```
pi@raspberrypi ~ $ touch test file
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwxr-xr-x 2 pi pi 4096 May 13 11:01 Downloads
-rw-r--r-- 1 pi pi 0 May 13 11:15 file
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:15 test
-rw-r--r-- 1 pi pi 0 May 13 11:10 testfile
-rw-r--r-- 1 pi pi 0 May 13 11:12 Testfile
pi@raspberrypi ~ $
```

STEP 6

Here are some other files names to avoid:

`##%&{} \<>*/$!":@+`|=`. The full stop (.) is used to create an extension to a file; usually used to indicate a file type, such as `textfile.txt` or `compressedfile.zip`, and starting a file with a full stop makes it invisible. Don't use a full stop in place of a space though; stick to underscores.

```
pi@raspberrypi ~ $ touch don't.use{odd}symbols&in<filenames>or=you'11^confi
```

REMOVING FILES

We've created some files that we don't want, so how do we go about removing them? It turns out that deleting files in your Raspberry Pi is really easy, which may be a problem, so be careful.

STEP 1

Enter: `ls -l` to view the files in your home directory. If you've followed the steps before then you should have three files: "test", "testfile", and "Testfile". We're going to get rid of these items because they were created as an example.

```
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwxr-xr-x 2 pi pi 4096 May 13 11:01 Downloads
-rw-r--r-- 1 pi pi 0 May 13 11:15 file
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:15 test
-rw-r--r-- 1 pi pi 0 May 13 11:10 testfile
-rw-r--r-- 1 pi pi 0 May 13 11:46 Testfile
pi@raspberrypi ~ $
```

STEP 2

To get rid of files you use the "rm" command. Enter: `rm Testfile` to delete the file called "Testfile" (with the uppercase "t"). Enter: `ls -l` and you'll find it's gone. Where is it? It's not in the Trash or Recycle Bin, like on a Mac or Windows PC. It's deleted completely and cannot be recovered. Bear this in mind and always think before deleting files.

```
pi@raspberrypi ~ $ rm Testfile
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 May 13 10:57 Documents
drwxr-xr-x 2 pi pi 4096 May 13 11:01 Downloads
-rw-r--r-- 1 pi pi 0 May 13 11:15 file
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 11:15 test
-rw-r--r-- 1 pi pi 0 May 13 11:10 testfile
pi@raspberrypi ~ $
```

STEP 3

We're going to use a wildcard (*) to delete our next two files, but again this is something you really need to do with care. First use "ls" to list the files and make sure it's the one you want to delete. Enter: `ls test*` to view files that match the word "test" and any other characters. The "*" character is called a "wildcard" and it means any characters here.

```
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Desktop
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Documents
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Downloads
-rw-r--r-- 1 pi pi 0 Jul 9 08:37 file
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 indiecity
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Scratch
-rw-r--r-- 1 pi pi 0 Jul 9 08:37 test
-rw-r--r-- 1 pi pi 0 Jul 9 08:37 testfile
pi@raspberrypi ~ $ ls test*
test testfile
pi@raspberrypi ~ $
```

STEP 4

We see that "ls test*" matches two files: "test" and "testfile", but not the file called "file". That's because it didn't match the "test" part of "test*". Check carefully over groups of files you want to remove (remember you can't recover them) and replace the "ls" with "rm". Enter: `rm test*` to remove both files. Finally enter: `rm file` to get rid of the confusing file.

```
pi@raspberrypi ~ $ rm test*
pi@raspberrypi ~ $ ls -l
total 24
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Desktop
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Documents
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Downloads
-rw-r--r-- 1 pi pi 0 Jul 9 08:37 file
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 indiecity
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 Jul 9 08:36 Scratch
pi@raspberrypi ~ $ rm file
pi@raspberrypi ~ $
```




Create and Remove Directories

Creating, moving and deleting directories isn't as easy in the Terminal as it is within a desktop interface. You need to tell Linux to move the directories inside other directories, a process known as recursion. Sounds complex but you should quickly get the hang of it.

MANAGING FILES AND DIRECTORIES

Now that you know how to create files, you'll want to learn how to make directories, which are the same thing as folders, as well as move items around. If you are more used to working with a desktop interface, this can take a bit of getting used to.

STEP 1

Enter: `ls` to quickly view all the directories currently in in the home location. Directories are created using the "mkdir" command (make directory). Enter: `mkdir testdir` to create a new directory in your home directory. Enter: `ls` again to see it.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ mkdir testdir
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ _
```

STEP 2

The "mkdir" command is different to touch, in that it doesn't update the timestamp if you use it with a directory that already exists. Enter: `mkdir testdir` again and you'll get the error "mkdir: cannot create directory 'testdir': File exists".

```
pi@raspberrypi ~ $ mkdir testdir
mkdir: cannot create directory 'testdir': File exists
pi@raspberrypi ~ $ _
```

STEP 3

Like touch, you can create multiple directories at once with the mkdir command. Enter: `mkdir testdir2 testdir3` and enter: `ls`. You'll now find several directories called testdir. Also, like files, you should know this means you can't (and really shouldn't) create directories with spaces. As with files, use an underscore ("_") character instead of a space.

```
pi@raspberrypi ~ $ mkdir testdir2 testdir3
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ _
```

STEP 4

You can create directories inside of each other using the directory path. Enter: `mkdir Documents/photos` to create a new directory called "photos" inside your documents directory. The directory has to already exist, though, try to enter: `mkdir articles/reports` and you'll get an error because there is no articles directory.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ mkdir Documents/photos
pi@raspberrypi ~ $ mkdir articles/reports
mkdir: cannot create directory 'articles/reports': No such
pi@raspberrypi ~ $
```

**STEP 5**

To create a directory path you need to pass in the “p” option to `mkdir` (which stands for “parents”). Options, if you remember, come after the command and start with a ‘-’. So enter: `mkdir -p articles/reports`. Enter: `ls` to view the articles directory, or “`ls articles`” to view the reports directory sitting inside.

```
pi@raspberrypi ~ $ mkdir -p articles/reports
```

STEP 6

Now you’re starting to get a bit more advanced, we’re going to just reiterate something. In Linux the command structure is always: command, option and argument, in that order. The command is the function, next are the options (typically single letters starting with “-”) and finally the argument (often a file, or directory structure). It’s always command, option then argument.

```
pi@raspberrypi ~ $ ls -l articles
total 4
drwxr-xr-x 2 pi pi 4096 May 13 12:36 reports
pi@raspberrypi ~ $ _
```

GETTING RID OF DIRECTORIES

Deleting directories is pretty easy in Linux, along with files, and this can be a problem. It’s too easy to delete entire directories containing files and these are instantly removed, not sent to a trash directory. Tread carefully.

STEP 1

We’re going to remove one of the directories we created earlier using the “`rmdir`” command. Enter: `ls` to view the files and directories in the current directory. We’ll start by getting rid of one of the test directories. Enter: `rmdir testdir3` and `ls` again to confirm the directory has been removed.

```
pi@raspberrypi ~ $ ls
articles Desktop Documents Downloads indiecity python
pi@raspberrypi ~ $ rmdir testdir3_
```

STEP 3

To delete a directory containing files or other directories, you return to the “`rm`” command used to remove files, only now we need to use the “-R” option (which stands for “recursive”). Using “`rm -R`” removes all the files and directories to whatever you point it at. Enter: `rm -R articles` to remove the articles directory.

```
pi@raspberrypi ~ $ ls
articles Desktop Documents Downloads indiecity python
pi@raspberrypi ~ $ rm -R articles
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $
```

STEP 2

Now we’ll try to get rid of the articles directory (containing the reports directory). Enter: `rmdir articles` and press return. You’ll get an error saying “`rmdir: failed to remove ‘articles’: Directory not empty`”. This is a puzzler; the `rmdir` command only removes directories that having nothing in them (no files or other directories).

```
pi@raspberrypi ~ $ rmdir articles
rmdir: failed to remove `articles': Directory not empty
pi@raspberrypi ~ $ _
```

STEP 4

As with multiple files, you can delete multiple directories inside the same directory using the “`rm`” command with the wildcard character (*). This should be done with care though so use the “-i” option (which stands for “interactive”). This will prompt you before each deletion. Enter: `rm -Ri test*` and press `Y` and `return` to each prompt. It’s a good idea to use the “-i” option whenever using the `rm` command.

```
pi@raspberrypi ~ $ rm -Ri test*
rm: remove directory `testdir'? y
rm: remove directory `testdir2'? y
rm: remove directory `testdir3'? y_
```



Copying, Moving and Renaming Files

Taking command of the Terminal is essential when learning how your Raspberry Pi's operating system works. The copying, moving and renaming of files is equally important, as you'll be doing a lot of this throughout your Pi projects.

USING THE MOVE COMMAND

In Linux, renaming a file is simply moving it from one name to another and copying a file is moving it without deleting the original. Don't panic, it's quite easy to master.

STEP 1

Before we can move anything around, we need to have a few test items in our home directory. Enter: `touch testfile` and `mkdir testdir` to create a test file and test directory in your home directory. Enter: `ls` to check that they are both present.

```
pi@raspberrypi ~ $ touch testfile
pi@raspberrypi ~ $ mkdir testdir
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
```

STEP 2

Files and directories are moved using the `mv` command. This is different to the commands we've looked at so far because it has two arguments (remember Linux command line is command, option, argument). The first argument is the source (the file or directory to be moved) and the second is the destination.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ mv testfile testdir
```

STEP 3

Enter: `mv testfile testdir` and press return to move the testfile document into the testdir directory. Enter: `ls` to see that it's no longer in the home directory, and `ls testdir` to see the testfile now sitting in the testdir directory. Now enter: `mkdir newparent` to create a new directory.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ mv testfile testdir
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ ls testdir
testfile
pi@raspberrypi ~ $
```

STEP 4

Directories with files are moved in the same way. Enter: `mv testdir newparent` to move the testdir directory inside the newparent directory. Let's move into the directory to find the file. Enter: `cd /newparent/testdir` and enter: `ls` to view the testfile sitting inside the directory.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity python_games Sc
pi@raspberrypi ~ $ mkdir newparent
pi@raspberrypi ~ $ mv testdir newparent
pi@raspberrypi ~ $ cd newparent/testdir
pi@raspberrypi ~/newparent/testdir $ ls
testfile
pi@raspberrypi ~/newparent/testdir $
```


**STEP 5**

Files and directories can be moved up using the double dot ("..") as an argument. Enter: `ls -la` to view your testfile and the single and double dot files. The single dot is the current directory and the double dot is the parent directory. Enter: `mv testfile ..` to move the testfile up into the newparent directory. Enter: `cd ..` to move up to the parent directory.

```
pi@raspberrypi ~ $ cd newparent/testdir
pi@raspberrypi ~/newparent/testdir $ ls
testfile
pi@raspberrypi ~/newparent/testdir $ mv testfile ..
pi@raspberrypi ~/newparent/testdir $ cd
```

STEP 6

You can also move files using longer paths. Enter: `cd ~` to return to the home directory and `mv newparent/testfile newparent/testdir/testfile` to move the testfile from its current location back inside the testdir directory. Enter: `ls newparent/testdir` to view the file back in its current directory.

```
pi@raspberrypi ~/newparent $ cd ~
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity newparent python
pi@raspberrypi ~ $ mv newparent/testfile newparent/testdir
pi@raspberrypi ~ $ ls newparent/testdir
testfile
pi@raspberrypi ~ $ _
```

RENAMING FILES AND DIRECTORIES

The `mv` command isn't used just to move files; it also serves the purpose of renaming files (effectively it moves it from its old name to a new name). Let's see how to use `mv` to rename items.

STEP 1

Let's start by making a new test file called "names". Enter: `touch testfile` and then `ls` to make sure the testfile is present. We're going to turn this into a file that contains the names of some people. So let's call it something more appropriate, like "names".

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity newparent python
pi@raspberrypi ~ $ mv testfile names
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity names newparent
pi@raspberrypi ~ $
```

STEP 3

You can rename directories inside other directories using paths. Let's rename the testdir directory, which is now inside the people directory. Enter: `mv names/testdir names/friends`. Now enter: `mv names people/friends` to move the names file inside the friends directory.

```
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity names people python
pi@raspberrypi ~ $ mv people/testdir people/friends
```

STEP 2

Enter: `mv testfile names` and `ls`. Now we can see the new "names" file in our directory. The `mv` command can also be used to rename directories. We should still have our newparent directory in our home directory. Enter: `mv newparent people` to rename the newparent directory. Enter: `ls` to view it.

```
pi@raspberrypi ~ $ touch testfile
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity newparent python
pi@raspberrypi ~ $ mv newparent people
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity people python_g
pi@raspberrypi ~ $
```

STEP 4

It is easy to overwrite files using the `mv` command, so if you have files with the same name use the "-n" option, which stands for "no overwrite". Enter: `touch testfile` to create a new file and `mv -n testfile people/friends`. There's no error report though, enter: `ls` and you'll find testfile still there.

```
pi@raspberrypi ~ $ touch testfile
pi@raspberrypi ~ $ mv -n testfile people/friends
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity people python_g
pi@raspberrypi ~ $ ls
Desktop Documents Downloads indiecity people python_g
pi@raspberrypi ~ $ ls people/friends
names testfile
pi@raspberrypi ~ $
```



Using the Man Pages

Linux comes with man (manual) pages that explain each command and show you all the options you can use. Once you get the hang of reading the man pages, you'll be able to find and do just about anything in Linux.

HEY, MAN!

The man pages are one of the best features of Linux, and as a built-in tool it's invaluable for both beginner and senior level Linux administrators. Let's see how it works

STEP 1

Linux has a built-in manual, known as man for short. Using the man command you can obtain information on all the Linux commands we've talked about. Simply enter: **man** and the name of the command you want to learn more about. Start by entering: **man ls** in the command line.

```
pi@raspberrypi ~ $ man ls
```

STEP 2

The man pages are a bit more detailed than you might be used to. First you have a name, which tells you what the command is called; in this case "list directory contents" and then the synopsis shows you how it works. In this case: "**ls** [OPTION].. [FILE..]". So you enter: **ls** followed by options (such as **-la**) and the file or directory to list.

```
LS(1)
NAME
  ls - list directory contents

SYNOPSIS
  ls [OPTION]... [FILE]...

DESCRIPTION
  List information about the FILES (the current directory by default). Sort entries alpha-
  betically by default. Mandatory arguments to long options are mandatory for short options too.

  -a, --all
      do not ignore entries starting with .

  -A, --almost-all
      do not list implied . and ..

  --author
      with -l, print the author of each file

  -b, --escape
      print C-style escapes for nongraphic characters

  --block-size=SIZE
      scale sizes by SIZE before printing them. E.g., '--block-size=M' prints sizes in
      units of 1,048,576 bytes.

  -B, --ignore-backups
      do not list implied entries ending with ~

  -c, --sort=TYPE
      with -lt: sort by, and show, ctime (time of last modification of file status info)
  -C, --sort=name
      list entries by columns

  --color[=WHEN]
      colorize the output. WHEN defaults to 'always' or can be 'never' or 'auto'. No
      effect if the output is not to a terminal.

  -d, --directory
      list directory entries instead of contents, and do not dereference symbolic links

  -D, --dired
      generate output designed for Emacs' dired mode

  -f, --full-time
      do not sort, enable -ad, disable -ls -color

  -F, --classify
      append indicator (one of */>@) to entries

  --file-type
      likewise, except do not append '*'

  --format=WORD
      across -x, commas -n, horizontal -x, long -l, single-column -l, verbose -l, vert
      ical -l

  --full-time
      like -l --time-style=full-iso

Manual page ls(1) line 1 (press h for help or q to quit).
```

STEP 3

Most commands are pretty easy to figure out how to use, so what you spend most of the time in the man pages is looking under the Description. Here you will see all the options and the letters used to activate them. Most man pages are longer than a single page, so press any key, such as the space bar, to move to the next page of content.

```
-g like -l, but do not list owner
--group-directories-first
  group directories before files.
  argument with a --sort option, but any use of --sort=name (-U) disables grouping
-G, --no-group
  in a long listing, don't print group names
-h, --human-readable
  with -l, print sizes in human readable format (e.g., 1K 234M 2G)
-si likewise, but use powers of 1000 not 1024
-H, --dereference-command-line
  follow symbolic links listed on the command line
--dereference-command-line-symlink-to-dir
  follow each command line symbolic link that points to a directory
--hide=PATTERN
  do not list implied entries matching shell PATTERN (overridden by -a or -A)
--indicator-style=WORD
  append indicator with style WORD to entry names: none (default), slash (-p), file-type (-
  l), none
-i, --inode
  print the index number of each file
-l, --ignore=PATTERN
```

STEP 4

Press the H key while looking at a man page to view the commands you can use to control the view. This is called the Summary of Less Commands (the less command is something we'll come to when we look at editing text). For now realise that you can move back and forward with Z and W. Press Q to quit this help screen and return to the man page.

```
SUMMARY OF LESS COMMANDS

Commands marked with * may be preceded by a number, N.
Notes in parentheses indicate the behavior if N is given.

h H          Display this help.
q |q Q |Q ZZ Exit.

MOVING

e ^E j ^N CR = Forward one line (or N lines).
y ^Y k ^K ^P = Backward one line (or N lines).
f ^F ^V SPACE = Forward one window (or N lines).
b ^B ESC-v    = Backward one window (or N lines).
z            = Forward one window (and set window to N).
u            = Backward one window (and set window to N).
ESC-SPACE    = Forward one window, but don't stop at end-of-file.
a ^P         = Forward one half-window (and set half-window to N).
u ^U         = Backward one half-window (and set half-window to N).
ESC-RightArrow = Left one half screen width (or N positions).
ESC-LeftArrow  = Right one half screen width (or N positions).
F             = Forward forever: like "tail -f".
r ^R ^L      = Repaint screen.
R            = Repaint screen, discarding buffered input.

Default "window" is the screen height.
Default "half-window" is half of the screen height.

SEARCHING

/pattern      = Search forward for (N-th) matching line.
?pattern      = Search backward for (N-th) matching line.
n            = Repeat previous search (for N-th occurrence).
N            = Repeat previous search in reverse direction.
ESC-n        = Repeat previous search, spanning files.
ESC-N        = Repeat previous search, reverse dir. & spanning files.
ESC-+        = Undo (toggle) search highlighting.
Apattern      = Display only matching lines

Search patterns may be modified by one or more of:
^N or ! Search for Nth-matching lines.
^E or * Search multiple files (pass thru END OF FILE).
^F or # Start search at FIRST file (for /) or last file (for ?).
^G Highlight matches, but don't move (KEEP position).
```



STEP 5

Scroll to the bottom of the man page to discover more information. Typically you will find the author's name and information on reporting bugs, including web links that can be useful for more information. Press Q to exit the man page and return to the command line.

```

assume tab stops at each COLS instead of 8
-u with -it: sort by, and show, access time with -l: show access time and sort by name ot
-U do not sort: list entries in directory order
-u natural sort of (version) numbers within text
-u, --width=COLS
    assume screen width instead of current value
-x list entries by lines instead of by columns
-X sort alphabetically by entry extension
-Z, --context
    print any SELinux security context of each file
-l list one file per line
--help display this help and exit
--version
    output version information and exit
SIZE may be (or may be an integer optionally followed by) one of following: KB 1000, K 1024, M

```

USING MAN OPTIONS

Because man doesn't change anything, like mv or mkdir, it is tempting not to see it as a command. But it is, and like all other commands it has options. These can be very handy to learn.

STEP 1

Entering: **man man** enables you to view some of the options, but sometimes you'll just want a quick overview. Fortunately man has a built-in help option that quickly lists the options. Press Q if you're in a man page and enter: **man -h** at the command line.

```

-c, --catman          print physical location of cat file(s)
-R, --recode=ENCODING used by catman to reformat out of date cat pages
                      output source page encoded in ENCODING
Finding manual pages:
-l, --locale=LOCALE   define the locale for this particular man search
-n, --system=SYSTEM   use manual pages from other systems
-N, --manpath=PATH     set search path for manual pages to PATH
-S, --s, --sections=LIST use colon separated section list
-e, --extension=EXTENSION limit search to extension type EXTENSION
-i, --ignore-case      look for pages case-insensitively (default)
-l, --match-case       look for pages case-sensitively
      --regex          show all pages matching regex
      --wildcard       show all pages matching wildcard
      --names-only     make --regex and --wildcard match page names only,
                      not descriptions
-a, --all              find all matching manual pages
-u, --update           force a cache consistency check
      --no-subpages    don't try subpages, e.g. 'man foo bar' -> 'man
                      foo-bar'
Controlling formatted output:
-P, --pager=PAGER      use program PAGER to display output
-r, --prompt=STRING     provide the 'less' pager with a prompt
-7, --ascii            display ASCII translation of certain latin1 chars
-E, --encoding=ENCODING use selected output encoding
      --no-hyphenation, --nh turn off hyphenation
      --no-justification, --nj turn off justification
-p, --preprocessor=STRING STRING indicates which preprocessors to run:
g = grep, r = refer, w = ugrind
      --troff          use groff to format pages
-T, --troff-device=DEVICE use groff with selected device
-H, --html[=BROWSER]   use www-browser or BROWSER to display HTML output
-X, --gxditview[=RESOLUTION] use groff and display through gxditview
                        (X11):
                        -X = -D75, -X100 = -DX100, -X100-12 = -DX100-12
-2, --ditroff          use groff and force it to produce ditroff
-?, --help             give this help list
--usage               give a short usage message

```

STEP 2

If you're fast you may have noticed the start of the text flew up off the page. This is because the "man -h" option doesn't use the less command by default (less is what enables you to move down text one screen at a time). We'll look into pipes ("|") later on, but for now just use "man -h | less" to read long text one page at a time.

```
pi@raspberrypi ~ $ man -h | less
```

STEP 6

The man command can be used for just about every command you use in Linux. You can even enter: man to get information on using the man tool. From now on, whenever you come across a new command in this book, such as "nano" or "chmod", take time to enter: **man nano** or **man chmod** and read the instructions.

```

MAN(1)

NAME
    man - an interface to the on-line reference manuals

SYNOPSIS
    man [-C file] [-d [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-n system],
    [-r prompt] [-?] [-E encoding] [--no-hyphenation] [--no-justification] [-p string] [-t],
    man -k [apropos options] regex ...
    man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
    man -f [whatis options] page ...
    man -i [-C file] [-d [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager],
    man -w|-W [-C file] [-d [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager] ...
    man -c [-C file] [-d [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager] ...
    man [-hw]

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is normally the name of a
    section, if provided, will direct man to look only in that section of the manual. The
    page found, even if page exists in several sections.

    The table below shows the section numbers of the manual followed by the types of pages that
    1 Executable programs or shell commands

```

STEP 3

One of the most powerful man options is the -k option, which is for "apropos". This enables you to search a wider range of man pages than the exact command. Enter: **man -k directory** to view all of the man pages relating to directories ("man -k directory | less" to view one page at a time). Here you'll find commands like "ls", "mkdir" and "cd" along with their description.

```

fchdir (2)          - change working directory
fchmodat (2)        - change permissions of a file relative to a directory file descriptor
fchownat (2)        - change ownership of a file relative to a directory file descriptor
fdunderdir (3)      - open a directory
find (1)            - search for files in a directory hierarchy
fstat (2)           - get file status relative to a directory file descriptor
fstat64 (2)         - get file status relative to a directory file descriptor
fstat (2)           - get current working directory
get_current_dir_name (3) - get current working directory
getcwd (2)          - get current working directory
getcwd (3)          - get current working directory
getdents (2)        - get directory entries
getdents64 (2)      - get directory entries
getdirenties (3)    - get directory entries in a file system-independent format
getuid (3)          - clone a repository into a new directory
git-clone (1)       - Move or rename a file, a directory, or a symlink
git-stash (1)       - Stash the changes in a dirty working directory away
help2man (1)        - generate the help tags file for directory
linkat (2)          - create a file link relative to directory file descriptors
lookup_cookie (2)   - return a directory entry's path
ls (1)              - list directory contents
mkdir (2)           - create a directory
mkdirat (2)         - create a directory relative to a directory file descriptor
mkdtemp (3)         - create a unique temporary directory
mkfifo (3)          - make a FIFO (named pipe) relative to a directory file descriptor
mkfontdir (1)       - create an index of X font files in a directory
mklost+found (8)    - create a lost+found directory on a mounted Linux second extended file system
mkmodat (2)         - create a special or ordinary file relative to a directory file descriptor
nktmp (1)           - create a temporary file or directory
modprobe.d (5)      - Configuration directory for modprobe
mountpoint (1)      - see if a directory is a mountpoint
oldfind (1)         - search for files in a directory hierarchy
openat (2)          - open a file relative to a directory file descriptor
opendir (3)         - open a directory
paw.mkdir (8)       - PAM module to create users home directory
pwd (1)             - print name of current/working directory
pwdx (1)           - report current working directory of a process
readdir (2)         - read directory entry
readdir_r (3)       - read a directory
readlinkat (2)      - read value of a symbolic link relative to a directory file descriptor
remove (3)          - remove a file or directory
renameat (2)        - rename a file relative to directory file descriptors
readdir (3)         - reset directory stream
rmdir (2)           - delete a directory
run-parts (8)       - run scripts or programs in a directory
scandir (3)         - scan a directory for matching entries
scandirat (3)       - scan a directory relative to a directory file descriptor

```

STEP 4

Entering the man page for all the commands you come across can be a little long-winded, although ultimately productive. If you simply want to know what a command does you can read just the description using the "whatis" command. Enter: **whatis pwd** to read the description of the "pwd" command ("print name of current/working directory").

```
pi@raspberrypi ~ $ whatis pwd
pwd (1)          - print name of current/working directory
pi@raspberrypi ~ $
```



Editing Text Files

A text file in Linux can be anything from a simple set of instructions on how to use an app, to some complex Python, C++ or other programming language code. Text files can be used for scripting, automated executable files, as well as configuration files too.

THE JOY OF TEXT

You will come across a lot of text files in Linux and to be able to edit or create a text file, you need a good text editor. Linux has many but here are some in action on the Raspberry Pi.

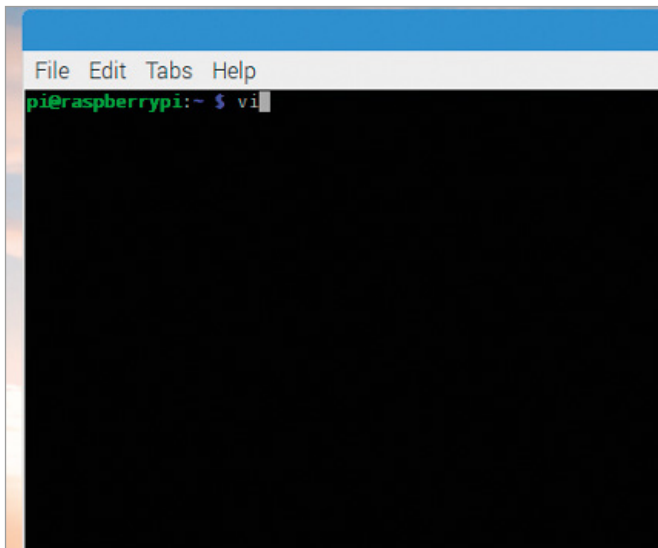
STEP 1

The first text editor for the Raspberry Pi is the default desktop environment app: Leafpad. To use, you can either double-click an existing text file or click the Raspberry Pi menu icon (in the top left of the desktop) and from the Accessories menu, choose Text Editor.



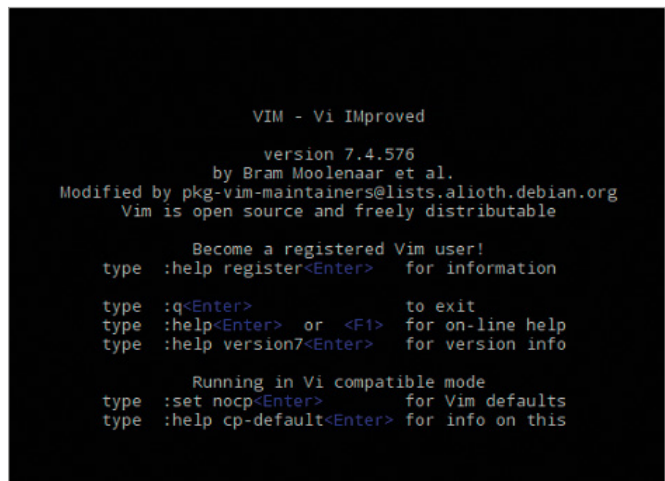
STEP 2

From the Terminal there are even more options, although using the correct command, you can launch any of the desktop apps via the Terminal. One of the simplest, and a classic text editor that's carried over from the Unix days, is vi. In the Terminal, enter: **vi**.



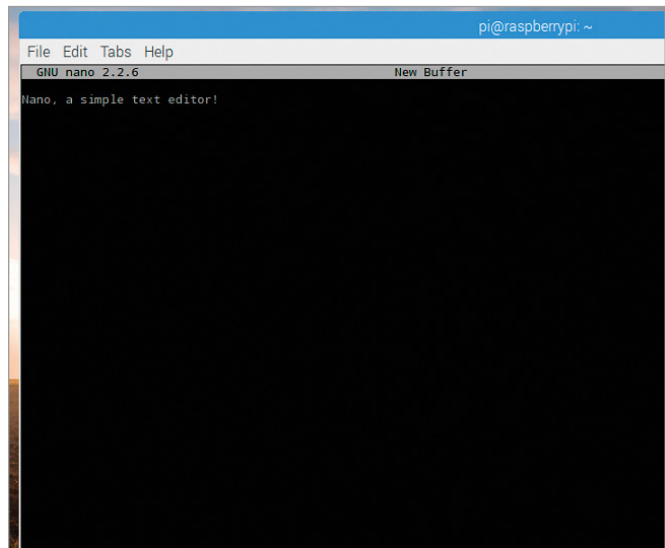
STEP 3

Vi is the original Unix command but in this case it launches VIM, the new Linux version of Vi. Although simple looking, Vi is considered, even by today's standards, to be one of the most widely used text editors. There's a lot you can do with it, so check out the man pages for more Vi information.



STEP 4

Nano is another favourite, and simple, text editor available for Linux. Enter: **nano** into the Terminal to launch it. You can use Nano for editing code, creating scripts or writing your own help files. To exit Nano, press Ctrl + X, followed by Y to save the file or N to exit without saving.





STEP 5

Emacs, or GNU Emacs, is an extensible and customisable, self-documenting, real-time display editor. It's a fantastic text editor and one that's worth getting used to as soon as you can. Sadly, it's not installed on the Pi by default, so you'll need to install it. In the Terminal, enter:

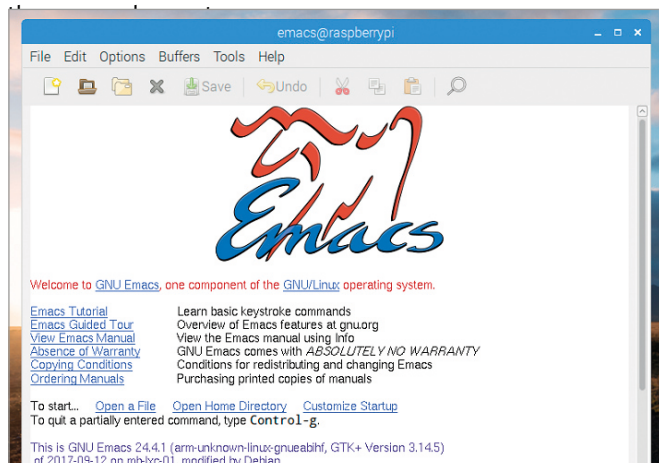
```
sudo apt-get install emacs
```

```
pi@raspberrypi: ~
File Edit Tabs Help

pi@raspberrypi:~$ sudo apt-get install emacs
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  emacs24 emacs24-bin-common emacs24-common ghostscript imagemagick-common libgs
  libm17n-0 libmagickcore-6.q16-2 libmagickwand-6.q16-2 libotf0 libpaper-utils
Suggested packages:
  emacs24-common-non-dfsg emacs24-el ghostscript-x m17n-docs libmagickcore-6.q16
  libm17n-0 libmagickcore-6.q16-2 libmagickwand-6.q16-2 libotf0 libpaper-utils
The following NEW packages will be installed:
  emacs emacs24 emacs24-bin-common emacs24-common ghostscript imagemagick-common
  libm17n-0 libmagickcore-6.q16-2 libmagickwand-6.q16-2 libotf0 libpaper-utils
0 upgraded, 18 newly installed, 0 to remove and 0 not upgraded.
Need to get 23.6 MB of archives.
After this operation, 105 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

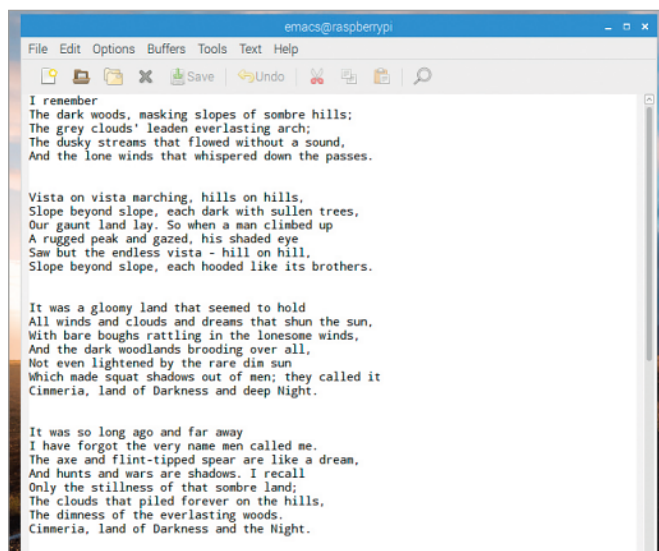
STEP 6

The previous command contacts the Debian (Raspberry Pi OS is based on a Debian Linux distribution) repositories and pulls down the information needed to install Emacs. When the Pi asks to continue with the installation, press Y. This installs the latest version and when it's done, you'll be back to



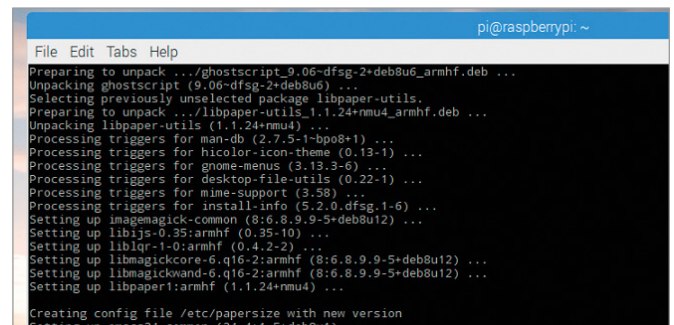
STEP 7

Once the installation is complete, enter: `emacs` into the Terminal. The Emacs splash screen opens in a new window, offering a tutorial (which we recommend you run through) and a guided tour amongst other information.



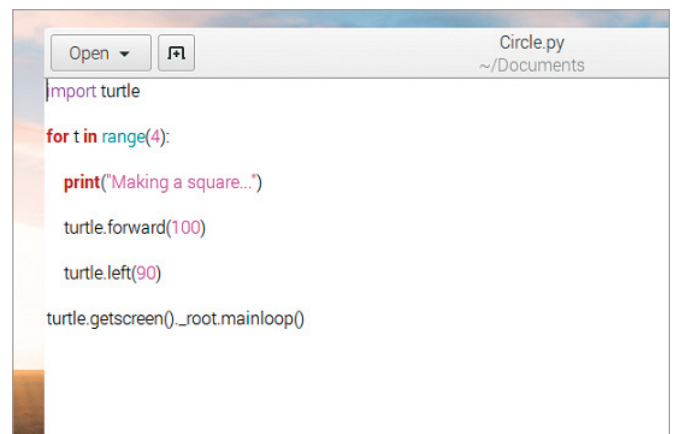
STEP 8

Emacs can offer an uncomplicated view of your text file or one with a plethora of information regarding the structure of the file in question; it's up to you to work out your own preference. There's also a hidden text adventure in Emacs, which we cover later in this book, why not see if you can find it without our help.



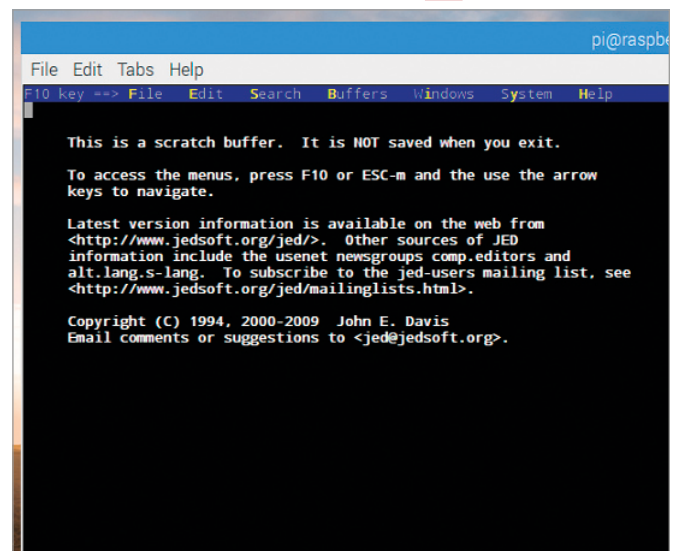
STEP 9

Gedit is another excellent text editor for Linux. Again, it's not installed by default on the Raspberry Pi; however, by entering: `sudo apt-get install gedit` and accepting the installation, the program can be on the Pi in a matter of seconds. Once it's installed, use `gedit` in the Terminal to launch it. Gedit is a great text editor for coding.



STEP 10

Finally, Jed is an Emacs-like, cross-platform text editor that's lightweight and comes with a wealth of features. To install it, enter: `sudo apt-get install jed`. Accept the installation and when it's complete, use: `jed` to launch.





Getting to Know Users

You might think you're the only person using your Raspberry Pi but there are several different users and even groups of users. Your main account is normally called Pi and there is an account above it called root, which is more powerful. You can also create users and groups.

WHAT IS A USER?

An important part of using Linux is the concept of users and understanding which user you are and which group you belong to. Like all modern computers, you can have multiple user accounts with each having different levels of access.

STEP 1

The first thing you need to do is get a concept of which user you are. Enter: `whoami` into the command line and press return. It should say "pi" (unless you set up your account name differently during setup). The "whoami" command might seem a bit simplistic, but it comes in very handy sometimes.

```
pi@raspberrypi ~ $ whoami
pi
pi@raspberrypi ~ $ _
```

STEP 2

When you are working in Linux, from time to time a 'Permission denied' error will occur, typically when you try to create, edit or execute (run) a file or directory outside of your area of privilege. If you want to see this, enter: `mkdir /testdir`. Attempting to create a new directory in your root directory isn't allowed.

```
pi@raspberrypi ~ $ mkdir /test
mkdir: cannot create directory '/test': Permission denied
pi@raspberrypi ~ $
```

STEP 3

To allow this, you need to use the sudo command. Sudo loosely stands for Substitute User Do; essentially it's the highest level of access to the system and you've already installed text editors using sudo. You'll come across sudo frequently in Linux, so let's create a second account to get the hang of it. Enter: `sudo useradd -m lucy` (or pick your name).

```
pi@raspberrypi ~ $ sudo useradd -m lucy
```

STEP 4

Now add a password for the new account. Enter: `sudo passwd lucy` and enter: a short password. Retype the same password and you'll now have two accounts on your Raspberry Pi. Now enter: `ls -l /home` to view the home directories for both users. Notice that the lucy directory lists lucy as the owner and group; and pi directory is belongs to pi.

```
pi@raspberrypi ~ $ sudo passwd lucy
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
pi@raspberrypi ~ $ ls -l /home
total 8
drwxr-xr-x  2 lucy  lucy  4096 May 13 19:01 lucy
drwxr-xr-x  3 pi    pi    4096 May 13 18:32 pi
pi@raspberrypi ~ $ _
```

**STEP 5**

Let's try switching to our new account. Enter: **su lucy** and enter the password you just created for that account. Notice that the command line now says "lucy@raspberrypi" but the working directory is "still /home/pi" (check this using "pwd"). Enter: **whoami** to confirm that you are now the new user.

```
lucy@raspberrypi /home/pi $ su lucy
Password:
lucy@raspberrypi /home/pi $ pwd
/home/pi
lucy@raspberrypi /home/pi $ _
```

STEP 6

We'll look at permissions in the next tutorial, but for now try to create a file as before. Enter: **touch testfile** to create a file. It will say "touch: cannot touch 'testfile': Permission denied". This is because your new user account doesn't have the right to create files in the /home/pi directory. Enter: **su pi** to switch back to your pi account.

```
pi@raspberrypi ~ $ su lucy
Password:
lucy@raspberrypi /home/pi $ touch testfile
touch: cannot touch 'testfile': Permission denied
lucy@raspberrypi /home/pi $ _
```

GETTING SUDO

We now have two accounts on our Raspberry Pi: lucy and pi. The lucy account can edit files in /home/lucy and the pi account can edit files in /home/pi. But there's also a third account, called "root", that sits above both lucy and pi. It can edit files anywhere.

STEP 1

The root account is all-powerful. It is possible, but not recommended, to switch to the root account, although you'll need to give it a password first (using "sudo passwd root"). Then just type "su" to switch to root. Please don't do this though: knowledge is a good thing but it's safer and wiser to use sudo instead.

```
pi@raspberrypi ~ $ sudo passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
pi@raspberrypi ~ $ su
Password: _
```

STEP 2

Most people think sudo stands for "super user", but it stands for "substitute user do". It enables you to perform a command as another user. Enter: **sudo -u lucy touch /home/lucy/test** to create a file inside the lucy home directory. You won't get an error because the lucy user has permission to edit that directory.

```
pi@raspberrypi ~ $ sudo -u lucy touch /home/lucy/test
pi@raspberrypi ~ $ _
```

STEP 3

It's rare that you use sudo to substitute another user. If you don't specify a user using the "-u" option with a username it defaults to the root account, as if you'd typed "sudo -u root". Enter: **sudo touch /home/lucy/anothertestfile** to create a file in the lucy directory while still using the pi account.

```
pi@raspberrypi ~ $ sudo touch /home/lucy/anothertest
pi@raspberrypi ~ $ ls /home/lucy/
anothertestfile  pstore.desktop  test
pi@raspberrypi ~ $ _
```

STEP 4

This step is optional. Only the pi user can use sudo. If we want to give the lucy account sudo privileges, it needs to be added to the sudoers file. Enter: **sudo visudo** to view the sudoers file. Add **lucy ALL=(ALL) NOPASSWD: ALL** to the last line and use Control+O to output the file. Remove the ".tmp" that is added to the file name as a security measure. Note that most accounts are not added to the sudoers file as a matter of course.

```
GNU nano 2.2.6 File: /etc/sudoers.tmp
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults    env_reset
Defaults    mail_badpass
Defaults    secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
# See sudoers(5) for more information on "include" directives:

#include /etc/sudoers.d
pi ALL=(ALL) NOPASSWD: ALL
lucy ALL=(ALL) NOPASSWD: ALL
```



Ownership and Permissions

Once you've got the hang of users, you need to learn about ownership and permissions. Different users have different areas of ownership and can do different things with each file. Permissions in Linux can be quite complex but with careful thought it's not too difficult.

OWNER AND PRIVILEGE

Each user account in Linux is an owner of a section of the filesystem: their Home area. Within this area, they do what they like (within reason), as they have owner privileges. Elsewhere though, they usually just have read-only privileges.

STEP 1

If you followed the previous tutorial you should now have two accounts on your Raspberry Pi. One called "pi" and the other with a name (Lucy in our case). An essential aspect of Linux is the idea of file and directory ownership; who owns, and has access, to what. You need a test file so enter: `touch testfile`.

```
pi@raspberrypi ~ $ touch testfile.txt
```

STEP 2

Now enter: `ls -l` and let's have a good look at the default permissions file. Our testfile.txt file starts with the text "-rw-r--r--". Start with the first letter, which is a dash '-'. All the other items in our home directory are directories. You can tell because they are blue, and our testfile.txt file is white.

```
pi@raspberrypi ~ $ touch testfile.txt
pi@raspberrypi ~ $ ls -l
total 28
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 3 pi pi 4096 May 13 18:08 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 3 pi pi 4096 May 13 14:53 people
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $
```

STEP 3

The first letter in the permissions also indicates a directory or file. Notice that all the other files start with a 'd' and our testfile.txt file starts with a '-'. That's what the first letter means. It's either a 'd', in which case it's a directory, or a '-', in which case it's not; it's a file. Enter: `ls -l testfile.txt` to view the permissions for just this file.

```
pi@raspberrypi ~ $ ls -l
total 28
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 3 pi pi 4096 May 13 18:08 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 3 pi pi 4096 May 13 14:53 people
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rw-r--r-- 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $ ls -l testfile.txt
-rw-r--r-- 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $
```

STEP 4

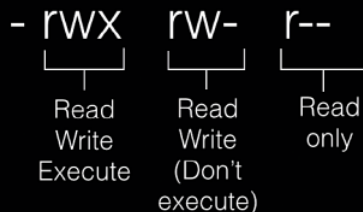
The next nine letters of the permissions are known as "alpha notation" because they let you know the permissions using letters. Each permission is either on, in which case you see a letter, or it is off, in which case you see a dash. The letter doesn't change for each place. So the first permission - the second letter after the directory one - is either an 'r' or a '-'. It's never any other letter.

```
pi@raspberrypi ~ $ ls -l testfile.txt
-rw-r--r-- 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $
```



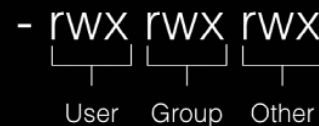

STEP 5

The 'r' means that particular permission is read and it's set to On. The nine letters here are divided into three groups of three letters: r, w, x. They stand for read, write and execute (run). Read means the file can be viewed (using cat or nano); w means the file can be edited or moved, and x means the file - typically a script or program - can be run.



STEP 6

The presence of r, w, or x means that this aspect is possible, a dash means it isn't. Our testfile.txt has no x letter; so if it were a script it wouldn't run. So why are there so many letters? Why not just three; read, write and execute? The three blocks of three letters are for different sets of people: user, group and other.



CHANGING PERMISSIONS

Now that you know how groups of permissions work, it's time to look at how to change them.

STEP 1

The first block of three is the most important. This is the user who owns the file (typically pi); the second is for other people in the same group as the user, and the third is for other people on the system. Permissions are changed using the chmod (change file mode bit) command. Enter: `man chmod` to look at the manual.

```
CHMOD(1)
NAME
    chmod - change file mode bits

SYNOPSIS
    chmod [OPTION]... MODE[,MODE]... FILE...
    chmod [OPTION]... OCTAL-MODE FILE...
    chmod [OPTION]... --reference=RFILE FILE...

DESCRIPTION
    This manual page documents the GNU version of chmod. chmod changes the file mode
    representing the bit pattern for the new mode bits.

    The format of a symbolic mode is [ugoa...][[-=][[perms...]]], where perms is ei
    rated by commas.

    A combination of the letters ugoa controls which users' access to the file will
    (a). If none of these are given, the effect is as if a were given, but bits that

    The operator * causes the selected file mode bits to be added to the existing fil
    except that a directory's unmentioned set user and group ID bits are not affected

    The letters rwxs select file mode bits for the affected users: read (r), writ
    for some user (X), set user or group ID on execution (s), restricted deletion fla
    granted to the user who owns the file (u), the permissions granted to other us
    gories (o).

    A numeric mode is from one to four octal digits (0-7), derived by adding up the b
    set group ID (2) and restricted deletion or sticky (1) attributes. The second d
    other users in the file's group, with the same values; and the fourth for other u
```

STEP 2

The chmod command is one of the trickier ones to understand. There are two ways you can adjust permissions; the first is using chmod with an option to target one of the three groups: owner, group, other. For these you use u, g or o followed by = and the letters or dashes you want. So enter: `chmod ugo=rwx testfile.txt` to make all three groups read, write and execute.

```
pi@raspberrypi ~ $ chmod ugo=rwx testfile.txt
pi@raspberrypi ~ $ ls -l
total 28
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 3 pi pi 4096 May 13 18:08 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 3 pi pi 4096 May 13 14:53 people
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rwxrwxrwx 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $
```

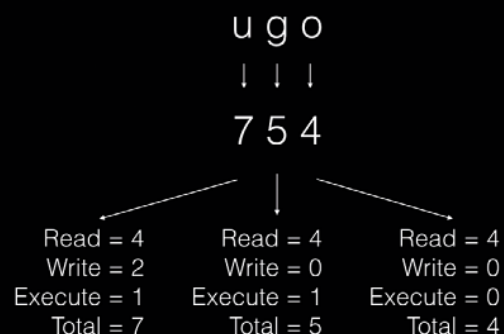
STEP 3

Turning everything on is probably overkill, so you need to target each group. Do this by putting commas between each mode option. Enter: `chmod u=rwx,g=rw,o=r testfile.txt` to give users read, write and execute privileges, user read and write and other just read. Enter: `ls -l` to see your handiwork.

```
pi@raspberrypi ~ $ chmod u=rwx,g=rw,o=r testfile.txt
pi@raspberrypi ~ $ ls -l
total 28
drwxr-xr-x 2 pi pi 4096 Apr 21 17:55 Desktop
drwxr-xr-x 3 pi pi 4096 May 13 18:08 Documents
drwx----- 2 pi pi 4096 May 13 11:01 Downloads
drwxr-xr-x 3 pi pi 4096 Apr 17 18:48 indiecity
drwxr-xr-x 3 pi pi 4096 May 13 14:53 people
drwxrwxr-x 2 pi pi 4096 Jan 1 1970 python_games
drwxr-xr-x 2 pi pi 4096 May 13 11:05 Scratch
-rwxrw-r-- 1 pi pi 0 May 13 21:14 testfile.txt
pi@raspberrypi ~ $
```

STEP 4

Alpha notation is fine, but many Linux admins use octal notation instead. This is a three-digit number that represents permissions. This is the formula: read=4, write=2 and execute=1 and you add them up for each group, therefore if a group is read, write and execute it's 7, if it's read and write it's 6 or if it's just execute it's 1. A popular option is 755. Enter: `chmod 755 testfile.txt` to change the file using octal notation.





Useful System and Disk Commands

Understanding these core Linux commands will enable you to not only master the inner workings of your Raspberry Pi but also to transfer those skills to other Linux distros, such as Ubuntu or Linux Mint.

LOTS OF LINUX

Linux is a huge and versatile command line language and there are hundreds of commands you can learn and use. Here are a few that can help you get more from your Raspberry Pi.

STEP 1

The Raspberry Pi is a great little computer, so let's start by getting some information. Enter: `cat /`

`proc/cpuinfo` to view some details on your Raspberry Pi processors. If you have a Raspberry Pi 3 you will see four processors, along with the model name and other info.

```
pi@raspberrypi ~ $ cat /proc/cpuinfo
processor       : 0
model name     : ARMv7 Processor rev 5 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
CPU implementer : 0xc11
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xc07
CPU revision   : 5

processor       : 1
model name     : ARMv7 Processor rev 5 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
CPU implementer : 0xc11
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xc07
CPU revision   : 5

processor       : 2
model name     : ARMv7 Processor rev 5 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae
CPU implementer : 0xc11
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xc07
CPU revision   : 5

processor       : 3
model name     : ARMv7 Processor rev 5 (v7l)
BogoMIPS      : 38.40
```

STEP 2

Remember that `cat` is used to list the contents of a text file, which is what `cpuinfo` is. There are other text files with system info available. Try "`cat /proc/meminfo`" to get information about your memory, "`cat /proc/partitions`" for information about your SD card, and "`cat /proc/version`" shows which version of Raspberry Pi you are using.

```
pi@raspberrypi ~ $ cat /proc/meminfo
MemTotal:       984304 kB
MemFree:        784576 kB
MemAvailable:   834088 kB
Buffers:        11660 kB
Cached:         56340 kB
SwapCached:      0 kB
Active:         49148 kB
Inactive:       30072 kB
Active(anon):   11256 kB
Inactive(anon): 228 kB
Active(file):   37892 kB
Inactive(file): 29044 kB
Unevictable:    0 kB
Mlocked:        0 kB
SwapTotal:     102396 kB
SwapFree:      102396 kB
Dirty:          36 kB
Writeback:      0 kB
AnonPages:     11132 kB
Mapped:        7464 kB
Shmem:         260 kB
Slab:          10304 kB
SReclaimable:  4512 kB
SUnreclaim:    5872 kB
KernelStack:   672 kB
PageTables:    564 kB
NFS_Unstable:   0 kB
Bounce:        0 kB
WritebackTmp:   0 kB
CommitLimit:   544548 kB
Committed_AS:  43712 kB
UmemTotal:     1171456 kB
UmemUsed:      3080 kB
UmemUsedChunk: 922368 kB
pi@raspberrypi ~ $ cat /proc/partitions
major minor #blocks name
179      0    7761920 mmcblk0
179      1    835890 mmcblk0p1
179      2         1 mmcblk0p2
179      3    32768 mmcblk0p3
179      5    61440 mmcblk0p5
179      6   6019040 mmcblk0p6
```

STEP 3

Enter: `uname` to view the name of the operating system's kernel, this is the element that sits between the interface and hardware. Just as you would suspect, the response from the command is Linux, as Raspberry Pi OS is a Linux distro, which in itself is based on another Linux distro called Debian. While it may sound complicated, it actually demonstrates how versatile Linux is.

```
pi@raspberrypi ~ $ uname
Linux
pi@raspberrypi ~ $
```

STEP 4

Enter: `uname -a` to view some more detailed information. Here you'll see the kernel name, hostname and kernel version (3.18.7-v7 on ours). If you have a Raspberry Pi 2 you'll see SMP (symmetric multiprocessing), followed by the system date, CPU architecture and operating system (GNU/Linux).

```
pi@raspberrypi ~ $ uname
Linux
pi@raspberrypi ~ $ uname -a
Linux raspberrypi 3.18.7-v7+ #755 SMP PREEMPT Thu Feb 12 2015
pi@raspberrypi ~ $
```

**STEP 5**

Enter: `vcgencmd measure_temp` to view the current operating system temperature of your Raspberry Pi.

Enter: `vcgencmd get_mem arm` to view the RAM available, and `vcgencmd get_mem gpu` to view the memory available to the graphics chip. Finally try `ls usb` to view a list of attached USB devices.

```
pi@raspberrypi ~ $ vcgencmd measure_temp
temp=36.9'C
pi@raspberrypi ~ $ vcgencmd get_mem arm
arm=880M
pi@raspberrypi ~ $ vcgencmd get_mem gpu
gpu=128M
pi@raspberrypi ~ $ lsusb
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 0449:1503 Holtek Semiconductor, Inc. Shortboard I
Bus 001 Device 005: ID 1a40:0101 Terminus Technology Inc. 4-Port HUB
Bus 001 Device 006: ID 276d:1105
pi@raspberrypi ~ $
```

STEP 6

One command you might be wondering about is how to switch off or restart your Raspberry Pi from the command line. Don't just hit the power switch. Enter: `sudo shutdown -h` now to shut down the Raspberry Pi (the "-h" option stands for "halt"), or enter: `sudo shutdown -r` now to restart your Raspberry Pi.

```
pi@raspberrypi ~ $ sudo shutdown -r now

Broadcast message from root@raspberrypi (tty1) (Thu May 14 12:20:29 2015):
The system is going down for reboot NOW!

-
```

DISK COMMANDS

Learn the two commands that enable you to view your disk space and the files on it: `df` (disk free space) and `du` (disk usage). With these two commands you can view the file usage on your SD card.

STEP 1

Start by entering: `df` in the command line. It returns a list of the volumes contained on your SD card. You

might be wondering what a volume is. It's best to think of your SD card as the drive. This contains partitions, which is where you split one drive to act like two or more drives. And each partition can contain volumes, which are storage spaces.

```
pi@raspberrypi ~ $ df
Filesystem      1K-blocks    Used Available Use% Mounted on
rootfs          6581636 3484164  2740096  56% /
/dev/root       6581636 3484164  2740096  56% /
devtmpfs        437856    0    437856    0% /dev
tmpfs           88432     260    88172    1% /run
tmpfs           5120      0    5120     0% /run/lock
tmpfs          176860    0   176860    0% /run/shm
/dev/mmcblk0p5  60479    14536   45943    25% /boot
pi@raspberrypi ~ $
```

STEP 2

Enter: `df -h` to get the list in human readable form. The first two lines should read "rootfs" and "/dev/

root" and have matching Size, Used, Avail and Use% listings. This is the main drive, and is an indication of how much space you have used, and have free, on your Raspberry Pi OS. The other volumes are for booting and initialising devices (you can ignore these for now).

```
pi@raspberrypi ~ $ du -h | less
22M  ../minecraft/games/com.mojang/minecraftWorlds/world
22M  ../minecraft/games/com.mojang/minecraftWorlds
22M  ../minecraft/games/com.mojang
22M  ../minecraft/games
4.0K  ../pulse
16K  ../config/gedit
8.0K  ../config/libfm
1.4M  ../config/epiphany/adblock
1.5M  ../config/epiphany
8.0K  ../config/lxsession/LXDE-pi
12K   ../config/lxsession
8.0K  ../config/dconf
8.0K  ../config/rncbc.org
8.0K  ../config/lxterminal
8.0K  ../config/uk.ac.cam.cl
8.0K  ../config/IndieCity
4.0K  ../config/enchant
8.0K  ../config/lxpanel/LXDE-pi/panels
16K   ../config/lxpanel/LXDE-pi
24K   ../config/lxpanel
28K   ../config/openbox
9.0K  ../config/window
```

STEP 3

Now enter: `du`. You should see lots of text fly up the screen. This is the disk usage for the files contained in your home directory and their sub-directories. As with `df`, it is better to use `du` with the "-h" option to humanise the output. If you want to slow down the output, you'll also need to pipe it through `less`. Enter: `df -h | less` to view the files and their respective usage one page at a time.

```
pi@raspberrypi ~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs          6.3G  3.4G  2.7G  56% /
/dev/root       6.3G  3.4G  2.7G  56% /
devtmpfs        428M    0  428M   0% /dev
tmpfs           87M   260K   87M   1% /run
tmpfs           5.0M    0   5.0M   0% /run/lock
tmpfs          173M    0   173M   0% /run/shm
/dev/mmcblk0p5  60M   15M   45M   25% /boot
pi@raspberrypi ~ $
```

STEP 4

You don't typically enter: `du` on its own; most of the time you want to view the disk usage of a specific directory. Enter: `du -h python_games` to view how much space the `python_games` directory (installed alongside Raspberry Pi OS) takes up. It should be 1.8M. If you want a more comprehensive breakdown of the files contained, use the "-a" option (all). Enter: `du -ha python_games` to view all the files contained and their disk usage.

```
pi@raspberrypi ~ $ ls
Desktop  Documents  Downloads  indicity  people  python_games  Scratch  test
pi@raspberrypi ~ $ du -h python_games
1.8M  python_games
pi@raspberrypi ~ $ du -ha python_games
12K  python_games/RedSelector.png
12K  python_games/4row_board.png
12K  python_games/Star.png
28K  python_games/4row_humanwinner.png
12K  python_games/Wall_Block_Fall.png
8.0K  python_games/princess.png
12K  python_games/Selector.png
8.0K  python_games/4row_black.png
4.0K  python_games/catanation.py
20K  python_games/flippy.py
36K  python_games/match3.wav
24K  python_games/starpusher.py
4.0K  python_games/grass1.png
40K  python_games/beep2.ogg
12K  python_games/slidepuzzle.py
8.0K  python_games/gen5.png
16K  python_games/fourinarow.py
8.0K  python_games/gen2.png
336K python_games/flippyboard.png
12K  python_games/Grass_Block.png
40K  python_games/match2.wav
```



Managing Programs and Processes

Being able to effectively manage the active programs and processes on your Raspberry Pi allows you to change the way the systems work. If you have a project requiring more memory, you can kill a process to free up the available system resources.

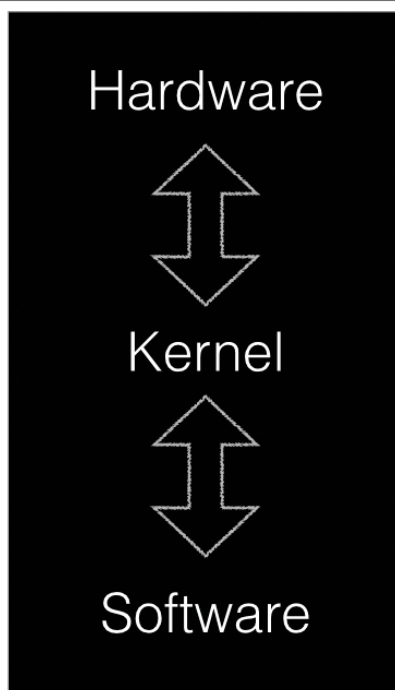
PROGRAMS AND PROCESSES

Linux has a trick up its sleeve when it comes to being able to manage programs and processes. When Windows closes a program, the allocated memory often isn't freed up again. However, Linux is far more streamlined.

STEP 1

As you get into Linux

you'll start to hear more about processes and another thing called the "kernel". The kernel sits beneath the software and hardware. It passes instructions to the hardware running processes, which takes up memory, and when the process is finished it closes it and reclaims the memory.



STEP 2

You're probably used to thinking in terms of programs and most OS's tend to keep processes out of sight. In

Linux on the other hand, you're right in at the deep end. A process is like a program, only it's a single task running on your computer.

Programs as you know them, may be a single process, or multiple processes working together. In Linux, you should learn to manage processes. This is done using the "ps" (process status) command.

```
pi@raspberrypi ~ $ ps_
```

STEP 3

If you type in "ps" on its own you don't see much. You should see two items: bash and ps. Bash (Bourne Again Shell) is the command line environment you are typing in, and ps is the command you just entered. If that seems a little light, that's because it is. These are just the processes owned by the user and are running in the foreground.

```
pi@raspberrypi ~ $ ps
  PID TTY          TIME CMD
 2165 tty1        00:00:01 bash
 2376 tty1        00:00:00 ps
pi@raspberrypi ~ $ _
```

STEP 4

If you want to see

processes used by other users (including those started by root) enter: **ps**

-a. The option stands for all users. This still isn't everything though because it doesn't include background processes. For this you can enter either "ps -A" or "ps -e". This will show you every process on the system including the background processes. You may need to pipe it through less using "ps -e | less".

```

  PID TTY          TIME CMD
   1 ?            00:00:02 init
   2 ?            00:00:00 kthreadd
   3 ?            00:00:00 ksoftirqd/0
   5 ?            00:00:00 kworker/0:0H
   6 ?            00:00:00 kworker/u0:0
   7 ?            00:00:00 rcu_preempt
   8 ?            00:00:00 rcu_sched
   9 ?            00:00:00 rcu_bh
  10 ?           00:00:00 migration/0
  11 ?           00:00:00 migration/1
  12 ?           00:00:00 ksoftirqd/1
  14 ?           00:00:00 kworker/1:0H
  15 ?           00:00:00 migration/2
  16 ?           00:00:00 ksoftirqd/2
  17 ?           00:00:00 kworker/2:0
  18 ?           00:00:00 kworker/2:0H
  19 ?           00:00:00 migration/3
  20 ?           00:00:00 ksoftirqd/3
  22 ?           00:00:00 kworker/3:0H
  23 ?           00:00:00 khelper
  24 ?           00:00:00 kdevtmpfs
  25 ?           00:00:00 netns
  26 ?           00:00:00 perf
  27 ?           00:00:00 khungtaskd
  28 ?           00:00:00 writeback
  29 ?           00:00:00 crypto
  30 ?           00:00:00 bioset
  31 ?           00:00:00 kblockd
  32 ?           00:00:02 kworker/1:1
  33 ?           00:00:00 rpciod
  34 ?           00:00:00 kswapd0
  35 ?           00:00:00 fsnotify_mark
  36 ?           00:00:00 nfsiod
  42 ?           00:00:00 kthrotld
  44 ?           00:00:00 UCHIQ-0

```




STEP 5

The `ps` command is one of the oldest known, and it has a lot of legacy. Because of this, it also has alternative options that can be used without the dash at the start. The most common way to use `ps` is to enter: `ps aux`. Piping this command through `less` is the best way to go about using `ps`. Enter: `ps aux | less` to view your processes.

USER	PID	%CPU	MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	2152	1364	T	S	12:21	0:00	init (2)
root	2	0.0	0.0	0	0	T	S	12:21	0:00	kthreadd
root	3	0.0	0.0	0	0	T	S	12:21	0:00	ksoftirqd/0
root	5	0.0	0.0	0	0	T	S	12:21	0:00	khavard
root	7	0.0	0.0	0	0	T	S	12:21	0:00	rcu_sched
root	8	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bh
root	9	0.0	0.0	0	0	T	S	12:21	0:00	rcu_preempt
root	10	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	11	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	12	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	13	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	14	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	15	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	16	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	17	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	18	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	19	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	20	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	21	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	22	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	23	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	24	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	25	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	26	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	27	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	28	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	29	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	30	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	31	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	32	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	33	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	34	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	35	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	36	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	37	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	38	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	39	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	40	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	41	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	42	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	43	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	44	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	45	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	46	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	47	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	48	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	49	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	50	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	51	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	52	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	53	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	54	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	55	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	56	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	57	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	58	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	59	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	60	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	61	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	62	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	63	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	64	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	65	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	66	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	67	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	68	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	69	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	70	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	71	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	72	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	73	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	74	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	75	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	76	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	77	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	78	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	79	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	80	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	81	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	82	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	83	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	84	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	85	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	86	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	87	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	88	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	89	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	90	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	91	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	92	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	93	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	94	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk
root	95	0.0	0.0	0	0	T	S	12:21	0:00	rcu_gp
root	96	0.0	0.0	0	0	T	S	12:21	0:00	rcu_bm
root	97	0.0	0.0	0	0	T	S	12:21	0:00	rcu_db
root	98	0.0	0.0	0	0	T	S	12:21	0:00	rcu_mb
root	99	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wb
root	100	0.0	0.0	0	0	T	S	12:21	0:00	rcu_wk

STEP 6

The “`ps aux`” command displays the processes in columns. The first is the User followed by the Process ID (PID), then we see the percentage of CPU and memory.

root	54	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/2
root	55	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/1
root	56	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	57	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	58	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	59	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	60	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	61	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	62	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	63	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	64	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	65	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	66	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	67	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	68	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	69	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	70	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	71	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	72	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	73	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	74	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	75	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	76	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	77	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	78	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	79	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	80	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	81	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	82	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	83	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	84	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	85	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	86	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	87	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	88	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	89	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	90	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	91	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	92	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	93	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	94	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0
root	95	0.0	0.0	0	0	T	S	12:21	0:00	thunderbolt/0



Input, Output and Pipes

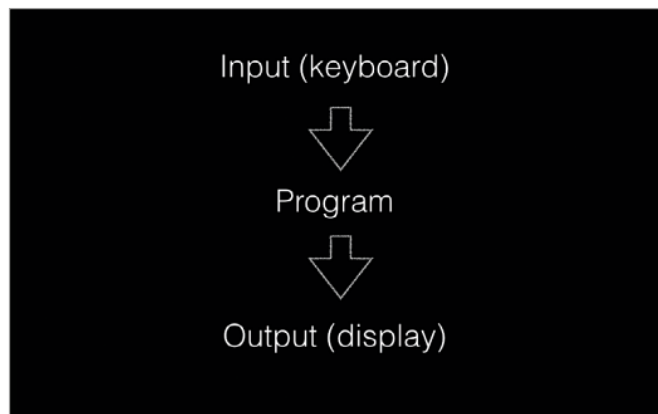
Most operating systems allow you to direct the output of something on the screen to, for example, a file. Linux, with its ancestral history of Unix-like command-based arguments, goes a step further and offers much more control.

IN, OUT, LINUX ALL ABOUT

Everything on a computer is about input and output. You input things into the computer (press keys, move the mouse) and it makes calculations and outputs content (changes the display, makes a noise, for example).

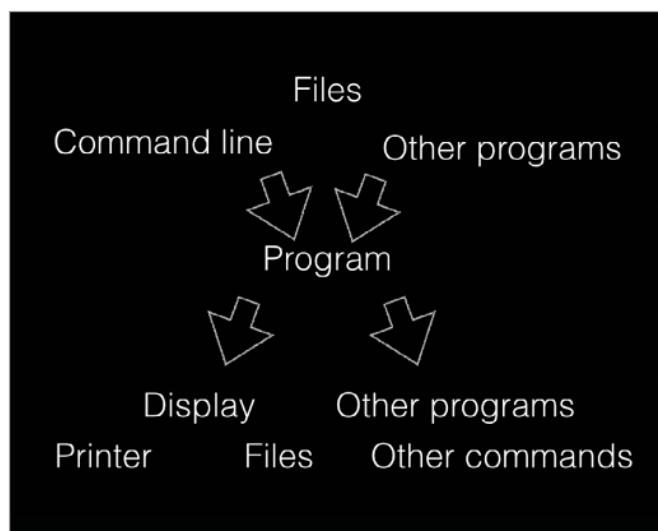
STEP 1

When you enter commands into Linux (in the command line), you are using standard input and output. This is so obvious that most people don't even think about it. You enter commands using the keyboard (that's input) and it responds to the screen (output). This is the regular way of doing it, which is why it's "standard input and output" (often called "stdin" and "stdout" for short).



STEP 2

As far as the computer is concerned, input and output can be to and from a whole range of different sources that you might never even think about. A program can get input from other programs, from files stored on the hard drive and a whole host of other areas. It outputs back to the display line, but also to files, other programs and even other commands.



STEP 3

You can change the standard output to something else using the ">" character after your command. If we wanted to see all the items in the python_games directory we could use "ls -l python_games". Using "ls -l python_games > games.txt" outputs the list of items to a new text file called "games.txt".

```

pi@raspberrypi ~ $ ls -l python_games > games.txt
  
```

STEP 4

The games.txt file now contains the output from the ls command. You can check it using "nano games.txt". It's an editable text file containing all the permissions, user and file size information and the names of the files. The output from the ls -l command, normally displayed on the screen, was instead sent to this file. Press Control+X to quit nano.

```

GNU nano 2.2.6
total 1800
-rw-rw-r-- 1 pi pi 9731 Jan 27 08:34 4row_arrow.png
-rw-rw-r-- 1 pi pi 7463 Jan 27 08:34 4row_black.png
-rw-rw-r-- 1 pi pi 8666 Jan 27 08:34 4row_board.png
-rw-rw-r-- 1 pi pi 18933 Jan 27 08:34 4row_computerwinner.png
-rw-rw-r-- 1 pi pi 25412 Jan 27 08:34 4row_humanwinner.png
-rw-rw-r-- 1 pi pi 8562 Jan 27 08:34 4row_red.png
-rw-rw-r-- 1 pi pi 8912 Jan 27 08:34 4row_tie.png
-rw-rw-r-- 1 pi pi 36908 Jan 27 08:34 badswap.wav
-rw-rw-r-- 1 pi pi 39782 Jan 27 08:34 beep1.ogg
-rw-rw-r-- 1 pi pi 39284 Jan 27 08:34 beep2.ogg
-rw-rw-r-- 1 pi pi 38581 Jan 27 08:34 beep3.ogg
-rw-rw-r-- 1 pi pi 39214 Jan 27 08:34 beep4.ogg
-rw-rw-r-- 1 pi pi 308 Jan 27 08:34 blankpygame.py
-rw-rw-r-- 1 pi pi 6581 Jan 27 08:34 boy.png
-rw-rw-r-- 1 pi pi 1034 Jan 27 08:34 catanimation.py
-rw-rw-r-- 1 pi pi 7270 Jan 27 08:34 catgirl.png
-rw-rw-r-- 1 pi pi 12574 Jan 27 08:34 cat.png
-rw-rw-r-- 1 pi pi 1178 Jan 27 08:34 drawing.py
-rw-rw-r-- 1 pi pi 83036 Jan 27 08:34 flippybackground.png
-rw-rw-r-- 1 pi pi 340760 Jan 27 08:34 flippyboard.png
-rw-rw-r-- 1 pi pi 19479 Jan 27 08:34 flippy.py
-rw-rw-r-- 1 pi pi 13080 Jan 27 08:34 fourinarow.py
-rw-rw-r-- 1 pi pi 2134 Jan 27 08:34 gameicon.png
-rw-rw-r-- 1 pi pi 4382 Jan 27 08:34 gem1.png
-rw-rw-r-- 1 pi pi 5665 Jan 27 08:34 gem2.png
-rw-rw-r-- 1 pi pi 3454 Jan 27 08:34 gem3.png
-rw-rw-r-- 1 pi pi 4217 Jan 27 08:34 gem4.png
-rw-rw-r-- 1 pi pi 5507 Jan 27 08:34 gem5.png
-rw-rw-r-- 1 pi pi 1681 Jan 27 08:34 gem6.png
-rw-rw-r-- 1 pi pi 3132 Jan 27 08:34 gem7.png
-rw-rw-r-- 1 pi pi 22489 Jan 27 08:34 gengem.py
-rw-rw-r-- 1 pi pi 3009 Jan 27 08:34 grass1.png
-rw-rw-r-- 1 pi pi 3019 Jan 27 08:34 grass2.png
-rw-rw-r-- 1 pi pi 3009 Jan 27 08:34 grass3.png
-rw-rw-r-- 1 pi pi 3032 Jan 27 08:34 grass4.png
-rw-rw-r-- 1 pi pi 8244 Jan 27 08:34 Grass_Block.png
-rw-rw-r-- 1 pi pi 7186 Jan 27 08:34 horngirl.png
-rw-rw-r-- 1 pi pi 18855 Jan 27 08:34 inkspillologo.png
-rw-rw-r-- 1 pi pi 18739 Jan 27 08:34 inkspill.py
-rw-rw-r-- 1 pi pi 7855 Jan 27 08:34 inkspillresetbutton.png
  
```

**STEP 5**

So > enables you to output to files, but you can also get input from a file. Make a new directory called music ("mkdir music") and switch to it ("cd music"). Enter: **nano bands.txt** to create a new text file. Enter some band names and press Control+O to output the file. Press Control+X to quit nano.

```
GNU nano 2.2.6

The Beatles
The Who
The Kinks
Credence Clearwater_Revival
Jefferson Airplane
Aerosmith
ZZ Top
-
```

STEP 6

We're going to use this text file as input to the sort command. Enter: **sort < bands.txt** and the content from the text file is used as input to sort. Because the output isn't specified, it uses the standard output (the screen) but you use input and output together. Enter: **sort < bands.txt > bands_sorted.txt** to create a new file with the band names in order to switch back to your pi account.

```
pi@raspberrypi ~/music $ sort < bands.txt
Aerosmith
Credence Clearwater_Revival
Jefferson Airplane
The Beatles
The Kinks
The Who
ZZ Top
pi@raspberrypi ~/music $ sort < bands.txt > bands_
```

USING PIPES

As well as directing input and output to and from files, you can send the output from one command directly into another. This is known as piping, and uses the pipe character "|".

STEP 1

As you start to get more advanced in Linux, you begin to create more powerful commands, and one way you do this is by using the pipe character ("|"). Take some time to find this character if you haven't already: it usually sits above or to the left of the Return key on most keyboards.

```
pi@raspberrypi ~/music $ ps aux | less
```

STEP 2

We've used the pipe a few times in the book ("ps aux | less"), but you might not have understood what's actually happening. Enter: **cat bands.txt | wc**. The output from the cat command (the text inside the document) isn't displayed on the screen, instead it is piped into the wc (word count) function. This then tells us how many lines, words and characters are in the document.

```
pi@raspberrypi ~/music $ cat bands.txt | wc
 7      13      94
pi@raspberrypi ~/music $ _
```

STEP 3

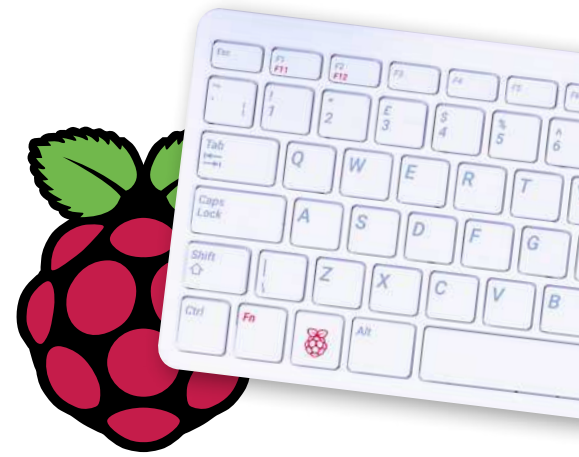
You can pipe commands multiple times. Enter: **cat bands.txt | sort | grep The*** to get the bands starting with "The" in alphabetical order. The output of the text from the bands.txt document is passed into sort, and the output from sort is passed into grep which filters out the bands starting with "The". These bands form the output.

```
pi@raspberrypi ~/music $ cat bands.txt | sort | grep The*
The Beatles
The Kinks
The Who
pi@raspberrypi ~/music $ _
```

STEP 4

You can combine pipes with input and output to create complex expressions. You can also use >> to append outputted data to a file that already exists. Enter: **cat bands.txt | wc >> bands.txt**. This takes the output from the bands.txt file and pipes it into the wc (word count) function. The output from wc is appended to the end of the bands.txt file. Enter: **cat bands.txt** to view it.

```
pi@raspberrypi ~/music $ cat bands.txt | wc >> bands.txt_
```



Command Line Quick Reference

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

TOP 10 COMMANDS

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

cd

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

mv

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

ls

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

chown

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

cp

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

chmod

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

pwd

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

rm

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

clear

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

mkdir

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



USEFUL HELP/INFO COMMANDS

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

free

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

df

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

top

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

uname

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

ps

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

grep

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

sed

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

adduser

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

deluser

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

delgroup

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

man man

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

man intro

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





Raspberry Pi Projects

The Raspberry Pi is all about projects. We've seen the Pi attached to a satellite taking high-resolution images of the Earth, as well as by scientists monitoring habitats in some of the most extreme places on the planet. And enthusiasts the world over have used a Raspberry Pi as the driving force of their unique projects.

Want to use the Pi to see where the International Space Station is? How about creating a retro coding system, or a media centre, or even connecting to a Bulletin Board System? You'll find these projects and more in this section. Where next is up to you. Let your imagination fly with the Raspberry Pi.

Creating a Loading Screen

If you're looking to add a little something extra to your Python code, then consider including a loading screen. The loading screen is a short introduction, or piece of art, that appears before the main part of your code.



LOADING

Back in the 80s, in the 8-bit home computing era, loading screens were often used to display the cover of a game as it loaded from tape. The image would load itself in, usually one-line-at-a-time, then proceed to colour itself in while the loading raster bars danced around in the borders of the screen.

Loading screens were a part of the package and often the buy-in

for the whole game as an experience. Some loading screens featured animations, or a countdown for time remaining as the game loads, while others even went so far as to include some kind of playable game. The point being: a loading screen is not just an artistic part of computing history, but an introduction to the program that's about to be run.

While these days loading screens may no longer be with us, in terms of modern gaming we can still include them in our own Python content. Either just for fun, or to add a little retro-themed spice to the mix.

SCREENS

Creating a loading screen in Python is remarkably easy. You have several options to hand: you can create a tkinter window and display an image, followed by a brief pause, before starting your main code, or you could opt for a console-based ASCII art version that loads one-line-at-a-time. Let's look at the latter and see how it works.

First you'll need some ASCII art, you can look up plenty of examples online, or use an image to ASCII Art converter to transform any images you have to ASCII. When you have your ASCII art, drop it into a newly created folder inside a normal text file.



Save the file, call it `screens.txt` for now.

THE CODE

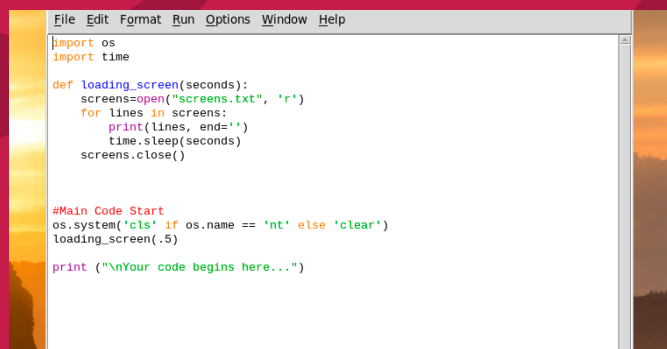
Launch Python and enter the following code to a New File:

```
import os
import time

def loading_screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()

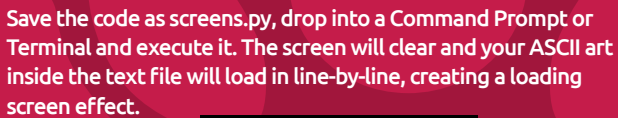
#Main Code Start
os.system('cls' if os.name == 'nt' else 'clear')
loading_screen(.5)

print ("\nYour code begins here...")
```



The code is quite simple: import the OS and Time modules and then create a Python function called `loading_screen` with a `(seconds)` option. Within the function, open the text file with the ASCII art as read-only and create a `for` loop that'll read the text file one-line-at-a-time. Next, print the lines – incidentally, the `lines, end=''` element will strip the newline from the text document, without it you'll end up with a double-line spaced display. Include the timing in seconds and close the text file buffer.

The final part of the code, `#Main Code Start`, is where you'll clear the screen (CLS for Windows, Clear for Linux (Raspberry Pi) and macOS) and call the function, together with the output number of seconds to wait for each line to be written to the screen – in this case, .5 of a second.



LOADING

79%

Create a New File in Python and enter the following code:

```
print ("\nYour code begins here...")
```

ADVENTURE TIME

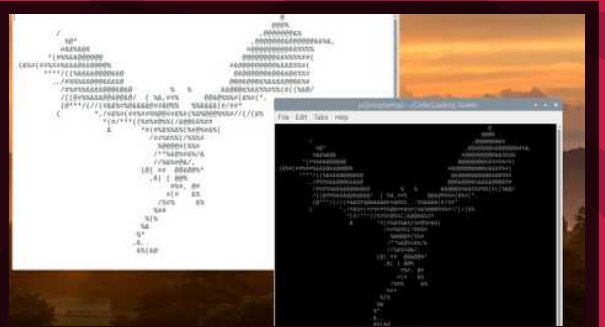
Imagine coming across a dragon, in game, and displaying its representation as ASCII. You can then load up the image lines, one-by-one, and continue with the rest of the adventure code. It's certainly worth having a play around with and it'll definitely add a little something else extra.

How about combining the two elements we've looked at? Let's begin with a Loading... progress bar, followed by the loading screen. After that, you can include your own code and continue your program. Here's the code:

```
loading_bar(10)
os.system('cls' if os.name == 'nt' else 'clear')
loading_screen(.5)
```

```
print ("\nYour code begins here...")
```

You can, of course, pull those functions up wherever and whenever you want in your code and they'll display, as they should, at the beginning. Remember to have the ASCII text file in the same folder as the Python code, or, at the `screens=open("screens.txt", 'r')` part of the code, point to where the text file is located.





Tracking the ISS with Pi

Of the many, amazing human achievements over the past decade or so, the International Space Station tops the bill. This incredible collaboration between nations sees vital experiments carried out in space, as well as observations of our own planet.

TO BOLDLY GO...

Indeed, the ISS is something most of us consider as a worthy example of what can happen when we work together. NASA, among other agencies, uses a wealth of Python code onboard the ISS to help automate routines, alongside its function as an in-between link to translate code from one language to another, and then into a human-readable format. If you're considering a career in space, then learning Python is a must-have skill.

While we're not able to fill you in on all the details regarding the code the ISS utilises, we can look at a fun Python script that will track the ISS, display the number of astronauts on board, and update the station's current latitude and longitude every five seconds, while also displaying your current latitude and longitude.

Displaying all that information in a single screen can become a little cluttered, so in this instance we're going to look at spreading all those details over three screens: a text document window - displaying the astronauts and your current latitude and longitude, a command line (or Terminal window) - displaying the continually updating latitude and longitude of the ISS as it orbits Earth, and a final, large window displaying a map of the world, together with an icon representing the ISS, that's updated as it orbits. Interested? Read on.





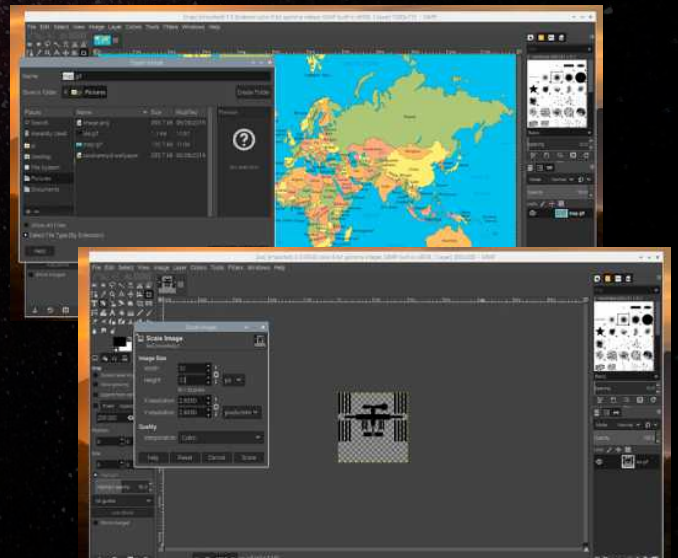
THE GRAPHICS

Firstly, we need to get hold of a map of the world and an image of the ISS, to use as an icon that will be updated according to the position of the space station as it travels over the surface. A quick Google of World Map will help you out here. Look for one that's reasonably large, the one we used for this example was 1280 x 700, and one that has the names of the countries – if you're using this with young people, to help with putting shapes of countries to names.



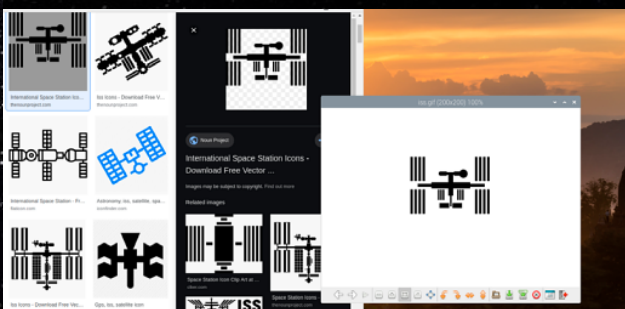
Next, look for an ISS icon. As this is going to be a graphical representation of the location of the ISS, we need the image to be reasonably small so it doesn't drown out the locations on the map, but also prominent enough to see when the map is loaded. We opted for an image that's 32 x 22 pixels in size. Don't worry too much if you're not able to find one that small, you can always resize it in an image-editing app such as GIMP.

As we're going to be using Turtle, a component of tkinter, the downloaded images will need to be converted to GIF, since this is the default and recommended image format. You can easily look up a converter online, but using GIMP, which is cross-platform and therefore works on both the Raspberry Pi and a Windows PC, will suffice. Simply load the image up in your image editor app and choose Save As, call them map and iss respectively, and click GIF as the image format. Remember to also resize the ISS image before saving it as a GIF.



THE CODE

The code we're using here utilises an open source API (Application Programming Interface) to retrieve real-time data online regarding the status of the ISS. An API enables applications to communicate with one another by providing the raw data that a programmer can pull out and interact with in their own code. In this case, the API in question is a web-based collection of raw data that's stored in a JSON (JavaScript Object Notation) format – an accessible and easy to use data-interchange interface.



In order for Python to interact with the JSON provided data, it needs to use the `urllib.request` and `json` modules. We're also going to be using a new module called `geocoder`, which will pull up a users' current latitude and longitude based on their IP address. The two JSON APIs can be located at: <http://api.open-notify.org/astros.json>, and, <http://api.open-notify.org/iss-now.json>. One of which contains the data regarding the astronauts onboard the ISS, and the other contains the data regarding the current location of the ISS in longitude and latitude.

```
message: "success"
number: 6
people:
  0:
    craft: "ISS"
    name: "Oleg Kononenko"
  1:
    craft: "ISS"
    name: "David Saint-Jacques"
  2:
    craft: "ISS"
    name: "Anne McClain"
  3:
    craft: "ISS"
    name: "Alexey Ovchinnikov"
  4:
    craft: "ISS"
    name: "Nick Hague"
  5:
    craft: "ISS"
    name: "Christina Koch"
```

```
Save Copy Collapse All Expand All
timestamp: 1557224376
message: "success"
iss_position:
  latitude: "24.8633"
  longitude: "-17.6535"
```


Let's begin by breaking the code into bite-sized chunks:

```
import json, turtle, urllib.request, time,
webbrowser

import geocoder # need to pip install geocoder
for your lat/long to work.
```

First, we need to import the modules used in the code: json, turtle, urllib.request, time, and webbrowser. The json and urllib.request modules deal with the data being pulled from the APIs, turtle will display the graphics, and time you already know. The Webbrowser module will open text files in the default browser or default text reader application. The geocoder module is a new element and you will need to install it with: `pip install geocoder` (or try `pip3 install geocoder`, if the pip install command still won't load the module). Geocoder can retrieve a users' location based on their IP address, as each ISP will have a geo-specific IP.

```
#Retrieve the names of all the astronauts
currently on board the ISS, and own lat/long -
write to a file and display

url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())

a=open("iss.txt","w")
a.write("There are currently " +
str(result["number"]) + " astronauts on the
ISS:\n\n")
```

```
people = result["people"]
```

```
for p in people:
    a.write(p["name"] + " - on board" + "\n")
```

```
g=geocoder.ip('me') # need to pip install
geocoder, and import as in the headers above
a.write("\nYour current Lat/Long is: " + str(g.
latlng)) # prints your current lat/long in the
text file.
```

```
a.close()
```

```
webbrowser.open("iss.txt")
```

This section will use the json and urllib.request modules to pull the data from the API that contains the names of the astronauts onboard the ISS. It then creates a new text file called iss.txt, where it'll write 'There are currently X astronauts on the ISS...' and list them by name. The second part of this section will then use the geocoder module to retrieve your current latitude and longitudes, based on your IP address, and also write that information to the end of the text file that contains the names of the astronauts. The last line, `webbrowser.open("iss.txt")` will use the webbrowser module to open and display the newly written text file in either the system's default text file reading app or the default web browser; either will work just fine.

```
#Setup world map in Turtle
```

```
screen = turtle.Screen()
```

```
screen.setup(1280, 720)
```

```
screen.setworldcoordinates(-180, -90, 180, 90)
```

```
#Load the world map image
```

```
screen.bgpic("map.gif")
```

```
screen.register_shape("iss.gif")
```

```
iss = turtle.Turtle()
```

```
iss.shape("iss.gif")
```

```
iss.setheading(45)
```

```
iss.penup()
```

This section of the code sets up the graphical window containing the world map and the ISS icon. To begin, we set up the turtle screen, using the resolution of the world map image we downloaded at the start of this tutorial (1280 x 720). The screen.setworldcoordinates syntax will mark the boundaries of the screen, creating the x and y coordinates of the four corners of the canvas, so that the ISS icon can freely travel across the map of the world and wrap itself back to the opposite side when it reaches an edge. The ISS icon is set as the turtle pen shape, giving it an angle of 45 degrees when moving.

```
while True:
```

```
    #Load the current status of the ISS in real-
    time
```

```
    url = "http://api.open-notify.org/iss-now.json"
```

```
    response = urllib.request.urlopen(url)
```

```
    result = json.loads(response.read())
```

```
    #Extract the ISS location
```

```
    location = result["iss_position"]
```

```
    lat = location["latitude"]
```

```
    lon = location["longitude"]
```

```
    #Output Latitude and Longitude to the console
```

```
    lat = float(lat)
```

```
    lon = float(lon)
```

```
    print("\nLatitude: " + str(lat))
```

```
    print("Longitude: " + str(lon))
```

```
    #Update the ISS location on the map
```

```
    iss.goto(lon, lat)
```

```
    #refresh every 5 seconds
```

```
    time.sleep(5)
```

Now for the final part of the code: Here we collect the location data from the ISS status API, pulling out the latitude and longitude elements of the JSON file. The code then prints the latitude and longitude data to the console/Terminal, transferring the data from a float to a string in order to print it correctly. The last section will use the latitude and longitude as variables lat and lon, to update the ISS icon on the map every five seconds.



Here's the code in its entirety:

```
import json, turtle, urllib.request, time, webbrowser
import geocoder # need to pip install geocoder for
your lat/long to work.

#Retrieve the names of all the astronauts currently on
board the ISS, and own lat/long - write to a file and
display
url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())
a=open("iss.txt","w")
a.write("There are currently " + str(result["number"])
+ " astronauts on the ISS:\n\n")

people = result["people"]

for p in people:
    a.write(p["name"] + " - on board" + "\n")

g=geocoder.ip('me') # need to pip install geocoder,
and import as in the headers above
a.write("\nYour current Lat/Long is: " + str(g.
latlng)) # prints your current lat/long in the text
file.
a.close()
webbrowser.open("iss.txt")

#Setup world map in Turtle
screen = turtle.Screen()
screen.setup(1280, 720)
screen.setworldcoordinates(-180, -90, 180, 90)
#Load the world map image
screen.bgpic("map.gif")

screen.register_shape("iss.gif")
iss = turtle.Turtle()
iss.shape("iss.gif")
iss.setheading(45)
iss.penup()

while True:
    #Load the current status of the ISS in real-time
    url = "http://api.open-notify.org/iss-now.json"
    response = urllib.request.urlopen(url)
    result = json.loads(response.read())

    #Extract the ISS location
    location = result["iss_position"]
    lat = location["latitude"]
    lon = location["longitude"]

    #Output Latitude and Longitude to the console
    lat = float(lat)
    lon = float(lon)
    print("\nLatitude: " + str(lat))
    print("Longitude: " + str(lon))

    #Update the ISS location on the map
    iss.goto(lon, lat)
    #refresh every 5 seconds
    time.sleep(5)
```

Create a new folder in your system, called ISSTrack (for example), and place the two graphics as well as the Python code itself.

```
File Edit Format Run Options Window Help

import json, turtle, urllib.request, time, webbrowser
import geocoder # need to pip install geocoder for your lat/long to work.

#Retrieve the names of all the astronauts currently on board the ISS, and own la
url = "http://api.open-notify.org/astros.json"
response = urllib.request.urlopen(url)
result = json.loads(response.read())
a=open("iss.txt","w")
a.write("There are currently " + str(result["number"]) + " astronauts on the ISS
people = result["people"]

for p in people:
    a.write(p["name"] + " - on board" + "\n")

g=geocoder.ip('me') # need to pip install geocoder, and import as in the headers
a.write("\nYour current Lat/Long is: " + str(g.latlng)) # prints your current la
a.close()
webbrowser.open("iss.txt")
|
#Setup world map in Turtle
screen = turtle.Screen()
screen.setup(1280, 720)
screen.setworldcoordinates(-180, -90, 180, 90)
#Load the world map image
screen.bgpic("map.gif")

screen.register_shape("iss.gif")
iss = turtle.Turtle()
iss.shape("iss.gif")
iss.setheading(45)
iss.penup()

while True:
    #Load the current status of the ISS in real-time
    url = "http://api.open-notify.org/iss-now.json"
    response = urllib.request.urlopen(url)
    result = json.loads(response.read())

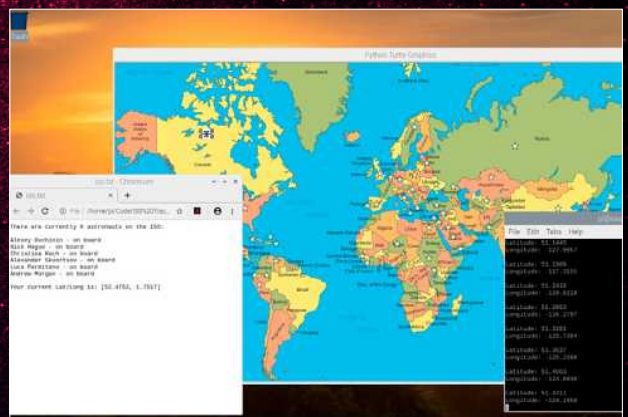
    #Extract the ISS location
    location = result["iss_position"]
    lat = location["latitude"]
    lon = location["longitude"]

    #Output Latitude and Longitude to the console
    lat = float(lat)
    lon = float(lon)
    print("\nLatitude: " + str(lat))
    print("Longitude: " + str(lon))

    #Update the ISS location on the map
    iss.goto(lon, lat)
    #refresh every 5 seconds
    time.sleep(5)
```

RUNNING THE CODE

The code is best executed from the command-line, or Terminal. Clear your desktop, enter your command line, and navigate to where you've saved the code plus the two graphics. Launch the code with either: `python3 ISSTrack.py`, or, `python ISSTrack.py` (depending on whether you're using a Raspberry Pi/Linux, or a Windows PC, and what you've called the Python code).



The code will launch two extra windows together with the command line window you already have open. One will be the text file, containing the named astronauts, along with your current latitude and longitude and the other will be the world map with the ISS icon located wherever the ISS is currently orbiting. The command line window will start scrolling through the changing latitude and longitude of the ISS.



ROCKET LAUNCH

Building on the previous countdown example, we can create an animated rocket that'll launch after the Blast Off!! message has been printed. The code would look something like:

```
def Rocket():
    distanceFromTop = 20
    while True:
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\n" * distanceFromTop)
        print("      /\      ")
        print("     ||      ")
        print("     ||      ")
        print("    /||\     ")
        time.sleep(0.2)
        os.system('cls' if os.name == 'nt' else 'clear')
        distanceFromTop -= 1
        if distanceFromTop < 0:
            distanceFromTop = 20

#Launch Rocket
Rocket()
```

#Launch Rocket
Rocket()

Here, we've created a new function called Rocket, which produces the effect of an ASCII-like rocket taking off and scrolling upwards; using the distanceFromTop variable.

To use this, add it to the end of the previous countdown code and, at the end of the Blast Off!! message, add the following lines:

```
print("\n\n\n\n")
input("Press Enter to launch rocket...")
```

This will allow your message to be displayed and then, when the user has hit the Enter button, the rocket will launch.

Again, the code in its entirety can be found in the Code Repository at: <https://bdmpublications.com/code-portal>.

```
File Edit Format Run Options Window Help
def Rocket():
    distanceFromTop = 20
    while True:
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\n" * distanceFromTop)
        print("      /\      ")
        print("     ||      ")
        print("     ||      ")
        print("    /||\     ")
        time.sleep(0.2)
        os.system('cls' if os.name == 'nt' else 'clear')
        distanceFromTop -= 1
        if distanceFromTop < 0:
            distanceFromTop = 20

#Launch Rocket
Rocket()
```

ROLLING DIE

Aside from the rocket animation, together with its countdown, another fun bit of text-based animation is that of a rolling dice.

A rolling dice can be a great animation to include in an adventure game, where the player rolls to see what their score is compared to that of an enemy. The highest roller wins the round and the losers' health drops as a result. It's an age-old combat sequence, used mainly in the Dungeon and Dragons board games and Fighting Fantasy novels, but it works well.

The code you'll need to animate a dice roll is:

```
import os
import time
from random import randint

die = ["  \n O \n  "] #1
die.append(" O\n \nO ") #2
die.append("O \n O \n O") #3
die.append("O O\n \nO O") #4
die.append("O O\n O \nO O") #5
die.append("O O\nO O\nO O") #6
```

```
def dice():
    for roll in range(0,15):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\n")
        number = randint(0,5)
        print(die[number])
        time.sleep(0.2)
```

#Main Code Begins
dice()

You may need to tweak the O entries, to line up the dots on the virtual dice. Once it's done, though, you'll be able to add this function to your adventure game code and call it up whenever your character, or the situation, requires some element of luck, combat, or chance roll of the dice.

```
File Edit Format Run Options Window Help
import os
import time
from random import randint

die = ["  \n O \n  "] #1
die.append(" O\n \nO ") #2
die.append("O \n O \n O") #3
die.append("O O\n \nO O") #4
die.append("O O\n O \nO O") #5
die.append("O O\nO O\nO O") #6

def dice():
    for roll in range(0,15):
        os.system('cls' if os.name == 'nt' else 'clear')
        print("\n")
        number = randint(0,5)
        print(die[number])
        time.sleep(0.2)

#Main Code Begins
dice()
```

DISCOVER ANIMATIONS

The great thing about Python code is that it's so accessible. These few examples will help you add some fun, or something different, to your programs, but they're just the tip of the proverbial iceberg. If there's something you want to include in your code, and you're at a sticking point, then consider heading over to Stack Overflow and search for Python 3 content.

Stack Overflow is a great online help and resource portal, where you can ask questions and experts will try and help you. It doesn't always work out, but most of the time, you'll find what you're looking for within this great resource.

Also, if you're after a simple animation then take to Google and spend some time searching for it. While you may not find exactly

what it is you're after, you're bound to come across something very like the desired effect; all you need to do is modify it slightly to accomplish your own goals.

You'll find that many professionals will often take to the Internet to find content they're after. True, they're able to code it themselves, but even experts get stuck sometimes, so don't worry about hunting code snippets down; you're in good company.

Home	5 votes	Q: How to create ASCII animation in a console application using Python 3.x?
PUBLIC	4 answers	I would like to port this question to Python (Windows + Linux + Mac: Os) How to create ASCII animation Windows Console application using C#? Thank you! ...
Stack Overflow		animation python-3.x console-application ascii-art asked Apr 28 '10 by Hamist Grubian
Tags		
Users	0 votes	Q: Simple rectangle terminal animation in Python 3.x
Julia		I have an assignment where I have to write a function that will print a rectangle (user can change it's height width) frame by frame with a square of "x" inside of it (I tried my best to descri



Retro Coding

There's a school of thought, that to master the foundations of good coding skills you need to have some experience of how code was written in the past. In the past is a bit of a loose term, but mostly, it means coding from the 80s.

```

5 HOME
10 PRINT "----UNIT CONVERTER----"
20 INPUT "CFJAH-CEL OR C2JEL-FA";C$
30 INPUT "ENTER UNIT: ";UN
40 X=(UNIT-32)*5/9
50 Y=(UNIT*9/5)+32
60 IF C$="F" THEN PRINT "CEL"
70 IF C$="C" THEN PRINT "FAH"

```

THE GOLDEN ERA OF CODE

While it may seem a little counterproductive to learn how to code in a language that's virtually obsolete, there are some surprising benefits to getting your hands dirty with a bit of retro coding. Firstly, learning old code will help you build the structure of code as, regardless of whether it is a language that was developed yesterday, or forty years ago, code still demands strict discipline to work correctly. Secondly, everyday coding elements, such as loops, sub routines and so on, are a great visual aid to learn in older code, especially BASIC. Lastly, it's simply good fun.

```

1050 REM FOR I=DLSTART TO BLEND
1060 REM PRINT I,PEEK(I)
1070 REM NEXT I
1080 REM
1090 POKE 512,0
1100 POKE 513,6
1110 REM
1120 FOR I=1536 TO 1550
1130 READ A

```

GOING BASIC

The easiest retro language to play around with is, without doubt, BASIC. Developed back in the mid-sixties, BASIC (Beginner's All-Purpose Symbolic Instruction Code) is a high-level programming language whose design was geared toward ease of use. In a time when computers were beginning to become more accessible, designers John Kemeny and Thomas Kurtz needed a language that students could get to grips with, quickly and easily. Think of BASIC as a distant relation to Python.

THE BEEB

The problem with BASIC is that there were so many different versions available, across multiple 8-bit platforms, with each having its own unique elements on top of the core BASIC functions. The

BASIC that was packaged with the Commodore 64 was different to that on the ZX

Spectrum, or the Atari home computers, due to the differing hardware of each system.

However, it's widely recognised that one of the 'best', and possibly most utilised, form of BASIC from the 80s was that of BBC BASIC.



BBC BASIC was used on the Acorn BBC Micro range of computers, utilising the MOS 6502-based processor technologies. It was one of the quickest examples of BASIC and, thanks to an inline assembler, it was also capable of allowing the developers of the time to write code for different processor types, such as the Zilog Z80 – a CPU present in the ZX Spectrum, as well as many arcade machines.

The BBC Micro was designed and built by Acorn Computers – a company that is historically responsible for the creation of the ARM CPU – the processor that's used in virtually every Android phone and tablet, smart TV, set top box and so on, as well as the Raspberry Pi – so essentially, the BBC Micro is the grandfather of the Raspberry Pi.

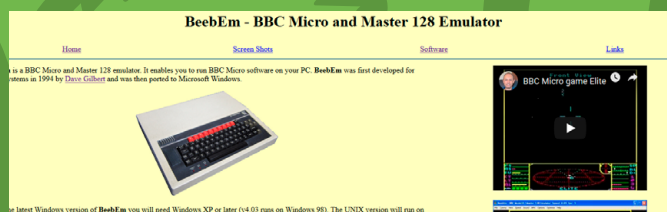
The BBC Micro was born in a time when the UK government was looking for a countryside computer platform to be used throughout education. Different companies bid, but it was the BBC's Computer Literacy Project (the BBC Micro) that was chosen, due to its ruggedness, upgradability, and potential for education. As a result, the BBC Micro, or the Beeb as it's affectionately known, became the dominant educational computer throughout the 80s.

BEEBEM

Naturally, you could scour eBay and look for a working BBC Micro to play around on, and it'll be a lot of fun. However, for the sake of just getting hands-on with some retro code, we'll use one of the best BBC Micro emulators available: BeebEm.

BeebEm was originally developed for UNIX in 1994 by Dave Gilbert and later ported to Windows. It is now developed by Mike Wyatt and Jon Welch, who maintain the Mac port of the emulator, and is therefore available for Windows 10, Linux and macOS, as well as other platforms.

If you're using Windows 10, simply navigate to <http://www.mkw.me.uk/beebem/index.html>, and download the BeebEM414.exe that's displayed in the main screen.



Once downloaded, launch the executable and follow the on-screen instructions to install it. MacOS users can get everything they need from: <http://www.g7jif.com/>. However, Raspberry Pi and Linux users will have to do a little nifty keyboard work before they can enjoy the benefits of the Beeb on their screens. Here's how to get it working under Linux:



First, drop to a Terminal and enter:

```
sudo apt-get update && upgrade
wget http://beebem-unix.bbcmicro.com/download/
beebem-0.0.13.tar.gz
```

Then, extract the compressed files with:

```
tar zxvf beebem-0.0.13.tar.gz, then enter the newly
created Beebem folder with: cd beebem-0.0.13/.
```

Now enter the following lines, hitting Enter and answering Y to accept any changes:

```
sudo apt-get install libgtk2.0-dev libsdl1.2-dev
./configure
```

```
make
```

```
sudo make install
```

This may take some time, but when it's all done, simply enter: beebem, to start the BBC Micro emulator.

BBC BASIC

Once installed and powered up, BeebEm will display the default BBC system start-up, along with a couple of beeps. Those of you old enough to have been in a UK school in the 80s will certainly recall this setup.

In BASIC, we use line numbers to determine which lines of code run in sequence. For example, to print something to screen we'd enter:

```
10 print "hello"
```

Once you've typed the above in, press Enter and then type:

```
run
```

We can of course expand the code to include variables, multi-line print statements, and so on:

```
1 CLS
```

```
10 Input "Hello, what's your name? " n$
```

```
20 print
```

```
30 print "Hi, " n$ " I hope
you're well today."
```

Type RUN to execute the code, you can also type LIST to view the code you've entered.

```
HI, DAVID I HOPE YOU'RE WELL TODAY.
>LIST
1 CLS
10 INPUT "HELLO, WHAT'S YOUR NAME? "
N$
20 PRINT
30 PRINT "HI, " N$ " I HOPE YOU'RE WE
LL TODAY."
>
```

As you can see, variables are handled in much the same way as Python, a print statement on its own displays a blank line, and CLS clears the screen; although the Pi uses Clear instead of CLS. We're also able to do some maths work, and play around with variables too:

```
1 CLS
```

```
10 input "how old are you? " a
```

```
20 print
```

```
30 if a > 40 print "You're over 40 years old."
```

```
40 if a < 40 print "You're under 40 years old."
```

```
YOU'RE OVER 40 YEARS OLD.
```

```
>LIST
1 CLS
10 INPUT "HOW OLD ARE YOU? " A
20 PRINT
30 IF A>40 PRINT "YOU'RE OVER 40 YEAR
S OLD."
40 IF A<40 PRINT "YOU'RE UNDER 40 YEA
RS OLD."
50 PRINT
>
```

```
50 print
```

BBC BASIC also has some interesting built-in features, such as the value of PI:

```
1 REM Area of a circle
```

```
2 CLS
```

```
20 Input "Enter the radius: " r
```

```
30 let area = PI*r*r
```

```
40 print "The area of your circle is: "; area
```

```
50 print ' '
```

```
>LIST
1 REM AREA OF A CIRCLE
2 CLS
20 INPUT "ENTER THE RADIUS: " R
30 LET AREA = PI*R*R
40 PRINT "THE AREA OF YOUR CIRCLE IS:
";AREA
50 PRINT ' '
>
```

As you'll notice, variables with a dollar (\$) represent strings, nothing after the variable, or a hash (#) represent floating point decimals; a whole integer has a % character, and a byte has an ampersand (&). The single quotes after the Print on line 50 indicate a blank line, one for each tick, while REM on line 1 is a comment, and thus ignored by the BASIC compiler.

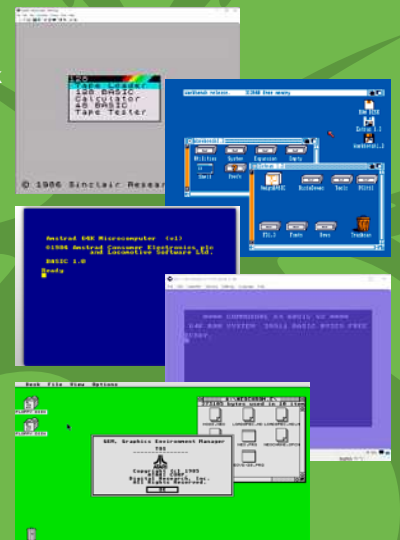


Needless to say, there's a lot you can learn, as well as having fun, with BBC BASIC. It's a rainy day project and something that's interesting to show the kids – this is how we rolled back in the 80s, kids!

There are a number of sites you can visit to learn BBC BASIC, such as <http://archive.retro-kit.co.uk/bbc.nvg.org/docs.php3.html>. See what you can come up with using BBC BASIC, or other BASIC types for different systems, and let us know what you've created.

OTHER SYSTEMS

Naturally, you don't have to look to the BBC Micro to play around with some retro code. If you grew up with a Commodore 64, then you can always try VICE, the C64 emulator. Likewise, the ZX Spectrum has a slew of great emulators available for every modern system to play around on. In fact, you can probably find an emulator for virtually every 8-bit or 16-bit machine that was produced over the years. Each has their own unique perspective and coding nuances, so find a few and see what you can create.



Using Text Files for Animation

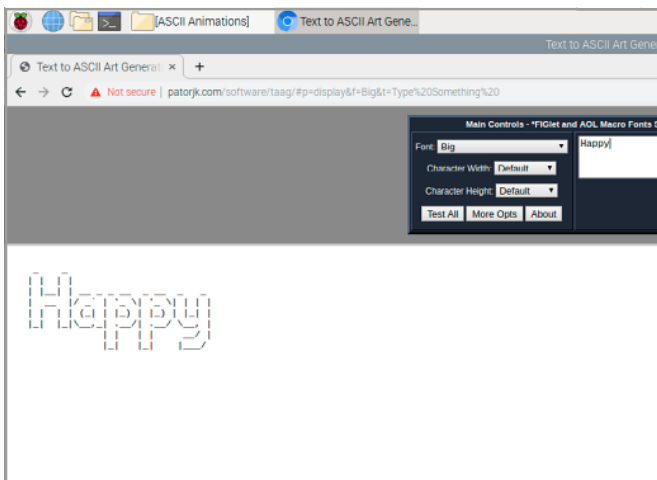
Animation in Python can be handled with the likes of the Tkinter and Pygame modules, however, there's more than one way to achieve a decent end result. Using some clever, text file reading code, we can create command-line animations.

ASCII ANIMATION

Let's assume you wanted to create an animated ASCII Happy Birthday Python script, with the words Happy and Birthday alternating in appearance. Here's how it's done.

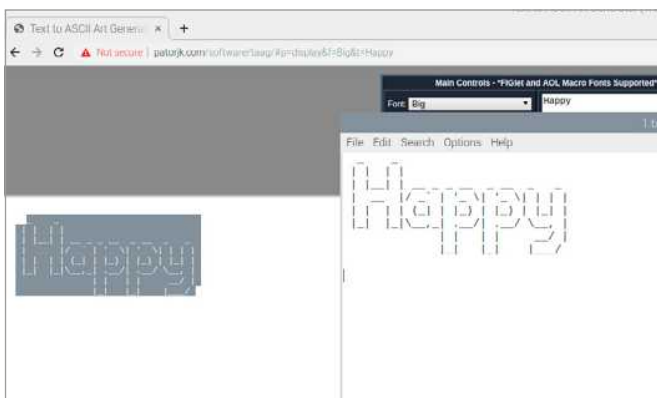
STEP 1

First we need to create some ASCII-like text, head over to <http://patorjk.com/software/taag>. This is an online Text to ASCII generator, created by Patrick Gillespie. Start by entering **Happy** into the text box, the result will be displayed in the main window. You can change the font with the drop-down menu to the side of the text box; we've opted for **Big**.



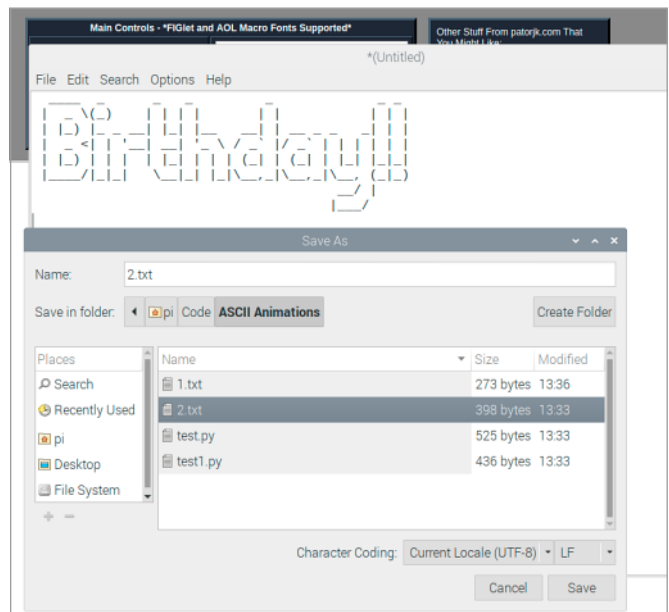
STEP 2

Now, on your computer create a folder in your Python code directory (call it **Test**, for now) and open either Leafpad for the Raspberry Pi or, if you're using Windows 10, then Notepad. Click on the **Select & Copy** button, at the bottom of the ASCII Generator webpage, and paste the contents into the text editor.



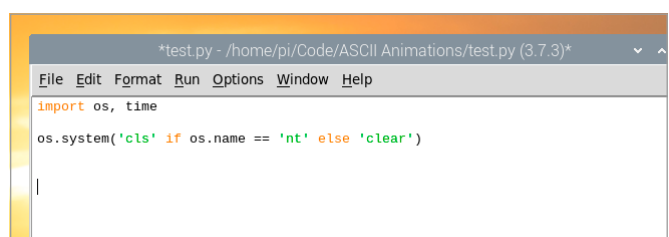
STEP 3

Save the text file as **1.txt** (you can call it anything, but now for ease of use 1.txt will suffice) and save the file in the newly created Test folder. When it's saved, do exactly the same for the word Birthday. You can select a new font from the ASCII Generator, or add extra characters, and when you're ready save the file as **2.txt**.



STEP 4

Open up Python and create a **New File**. We're going to need to import the OS and Time modules for this example, followed by a line to clear the screen of any content. If you're using Windows, then you'll use the CLS command, whereas it's Clear for the Raspberry Pi's Linux OS. We can create a simple if/else statement to handle the command.

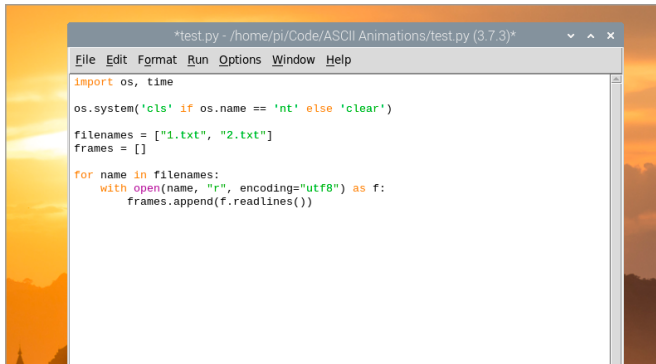


**STEP 5**

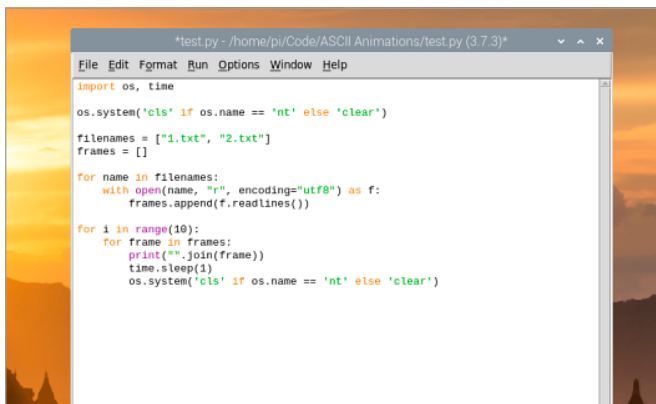
Next we need to create a list with the names of the text files we want to open, and then we need to open them for display in the Terminal.

```
filenames = ["1.txt", "2.txt"]
frames = []
```

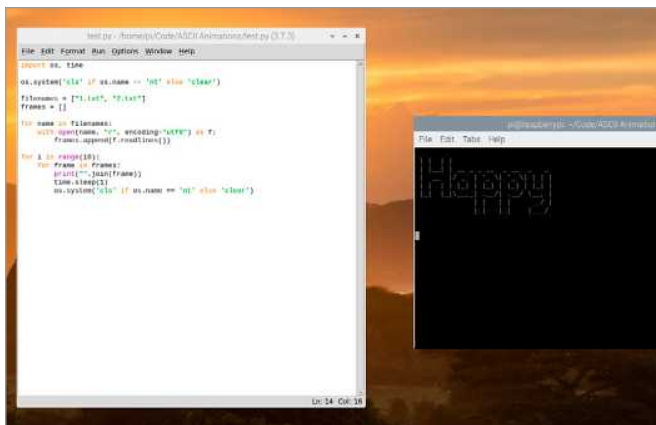
```
for name in filenames:
    with open(name, "r", encoding="utf8") as f:
        frames.append(f.readlines())
```

**STEP 6**

We've used the UTF8 standard when opening the text files, as ASCII art as text, within a text file, often requires you to save the file as UTF compliant – due to the characters used. Now we can add a loop to display the files as 1.txt, then 2.txt, creating the illusion of animation while clearing the screen after each file is displayed.

**STEP 7**

Save the Python code in the same folder as the text files and drop into a Terminal or Command Prompt. Navigate to the folder in question, and enter the command:
python3 NAME.py
Where NAME is whatever you called your saved Python code.

**STEP 8**

Here's the code in full:

```
import os, time
```

```
os.system('cls' if os.name == 'nt' else 'clear')
```

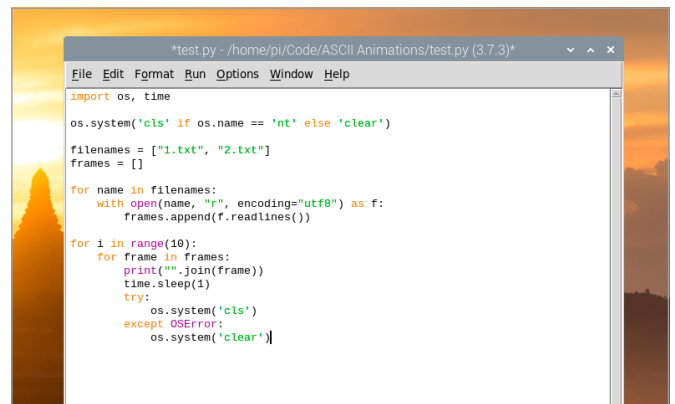
```
filenames = ["1.txt", "2.txt"]
frames = []
```

```
for name in filenames:
    with open(name, "r", encoding="utf8") as f:
        frames.append(f.readlines())
```

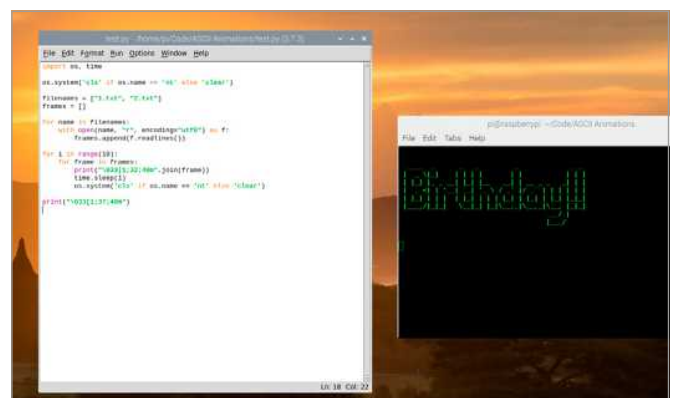
```
for i in range(10):
    for frame in frames:
        print("".join(frame))
        time.sleep(1)
    os.system('cls' if os.name == 'nt' else 'clear')
```

STEP 9

Note, from the loop, within the code, we've used the same CLS and Clear if/else statement as before. Again, if you're running on Windows then the OS module will use the CLS command, 'ELSE' if you're using Linux or a Mac, the Clear command will work correctly. If you want, you could use a Try/Except statement instead.

**STEP 10**

You can spice things up a little by adding system/ Terminal colours. You'll need to Google the system codes for the colours you want. The code in our example turns the Raspberry Pi Terminal to green text on a black background, then changes it back to white on black at the end of the code. Either way, it's a fun addition to your Python code.



Pi Projects: Desktop Pi

As you are now no doubt aware, the Raspberry Pi is certainly a versatile little piece of technology. Let's look at some of the popular projects you can apply to your Pi, starting with one that's not only the easiest, but also one of the best.

COMPUTER PI

First and foremost, the Raspberry Pi is a computer. It has an operating system, built-in productivity apps, the ability to connect to both a home network and the Internet (either wirelessly or through Ethernet), and you can connect a mouse, keyboard, and monitor. It stands to reason, then, that it makes for a remarkably cheap desktop computer.

The obvious advantage of using the Raspberry Pi as a desktop computer though, is the fact that it's so cheap. For around £100 you could easily purchase the Pi, monitor, keyboard and mouse, providing you with a fully-functional computer capable of doing most, if not all, of the things you would do on a computer that costs ten times that amount. However, there are ways in which you can further enhance the desktop Pi project.

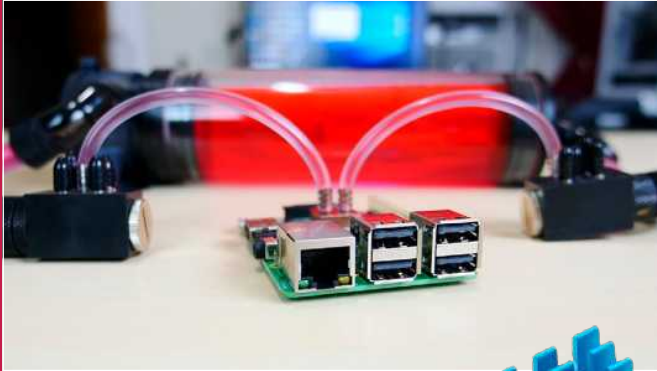


CASES

To begin with, you will need the Raspberry Pi OS Stretch with desktop and recommended software OS version from the Raspberry Pi Foundation page, [www.raspberrypi.org/downloads/Raspberry Pi OS](http://www.raspberrypi.org/downloads/Raspberry-Pi-OS). Once you've installed that to your Pi, booted it, and logged in with a personal user account, you can then start looking at one of the many colourful, and functional, cases available for the Pi.

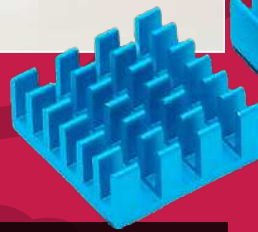
The Pi Hut offers a superb selection of cases for the various versions of the Raspberry Pi, for this project, we'd recommend the Raspberry Pi 3 Model B+, as it's the fastest and most capable of the current models. For just £6, you can have a colourful – white and red – official Raspberry Pi case, with pre-formed holes that line up perfectly with the ports on the Pi. For around ten pound more, there's also the FLIRC case, an impressive design that's finished in brushed aluminium. On the other hand, if you want something a little different from the norm, then how about a case that's designed to look like a Nintendo Entertainment System, a SNES, or even a SEGA Megadrive?





COOLING

Another thing to consider, if you're going to be using your Pi as a desktop computer, is cooling. While the Pi's low use of resources means that its internals don't get too hot, prolonged and intense use may see your Pi getting a little warm, especially if it's inside a case and the ambient temperature of the environment is also warm. There are options available to help cool your Pi down, chiefly the Raspberry Pi Heatsink. This is a high-quality aluminium heat dispersing element that fits on top of the Pi's processor, leeching heat away from the vital circuit board and out into the surrounding air. However, bigger and more effective heatsinks are available, as well as fans, and even a water-cooling kit. These are a little extreme though, and only necessary if your Pi is doing some serious number crunching for an extended period of time.



STORAGE

While the Raspberry Pi desktop project is a great idea, the sad fact of the matter is that the Pi doesn't exactly offer the user a wealth of storage; compared to a Windows or Mac desktop, that is. Naturally, you can use a larger capacity SD card on which to install Raspberry Pi OS, and utilise the remaining space as your Home folder, but again you're limited. Thankfully, it's not too much expense to include an external hard drive. A portable, USB connected, slimline 2TB hard drive can cost in the region of £60 and give you as much storage capacity as one of the more traditional desktop computers. One thing worth noting with regards to using an external hard drive, make sure you always have a backup of what's on there.

There are many options available to you if you decide to go down the Raspberry Pi desktop computer project route; there's even a Pi laptop kit available called Pi-Top. Needless to say, with this amount of choice you're able to customise your desktop and its setup in a more personal way than on a Windows PC, or a Mac.



PERIPHERALS

To cut down on the amount of cable clutter on your desktop, you could consider opting for a Bluetooth keyboard and mouse. The Pi 3 Model B+ offers Bluetooth connectivity alongside its Wi-Fi capabilities, so effectively you're able to use any recently purchased Bluetooth keyboard and mouse kit available. The major advantage here is that the Pi can be mounted to the rear of your monitor and, together with connecting to your home network via Wi-Fi, the only cables involved would be the power and the HDMI cable to the monitor, both of which could be neatly tucked away.



DON'T THROW AWAY YOUR PC YET

While the Raspberry Pi is a great little desktop computer, it does have its limitations. If you're accustomed to using a reasonably powerful Windows PC, or Mac, then you may find that, while being efficient, cheap and offering more space on your desk, the Pi does tend to struggle from time to time.

This is purely down to the fact that the Raspberry Pi isn't the most powerful computing device available today. It won't be able to handle the latest, triple-A games, intense graphics, Virtual Reality, or even some of the higher definition media content. You may find it stuttering when trying to playback fast moving 1080HD scenes, and true 4K playback isn't an option.

However, if you're looking to substitute your day-to-day workstation with something cheaper and smaller, and you're not going to push the Pi's processor too much, then you'll find the Pi to be a wonderful desktop computer. If you want more, though, hang on to that more powerful PC for the moment.

Pi Projects: BBS Client

In a digital world, before the Internet was a common household name, there existed a connected community of surfers. These individuals didn't surf the WWW, instead they dialled up Bulletin Board Systems (BBS) and opened a whole new world of content.

WARGAMES

If you're old enough to recall, or have since watched, the excellent movie *Wargames*, then you'll be roughly familiar with the way in which a Bulletin Board System works; and if you haven't watched *Wargames*, then we recommend you get hold of it.

In the movie, the young protagonist spends his days at the keyboard of his early 80s computer, using his modem to dial into remote systems. Once inside these remote systems, he then goes about traversing the remote host's file system looking for anything interesting.

The movie plot aside, this, essentially, is how a BBS works. It's a remote computer that runs specialist BBS server software with a mix of content either pre-installed, or added by the system admin (sysadmin or sysop). A user of the BBS can then dial, in the old modem sense, the BBS server's phone number and gain access to the system with a valid username and password; or if they're new, they have the option to create a new user.

These days, of course, the dial-up aspect has pretty much gone the way of the Dodo (although there are still some retro stalwarts who relish in the chronic noise of a dial-up connection), however, we can still enjoy the retro feeling of a traditional BBS using the legacy protocol, Telnet.



WHY?

In a world of Internet snooping, a BBS is probably one of the last bastions of digital privacy; to some degree. A private BBS is somewhere you can connect with likeminded individuals, to chat, swap code, reminisce, play a text-based adventure, or simply just hang out. True, you can get hold of copyrighted or explicit content, but that's only if you connect to those BBSes that serve such content, just as with the Internet.

Most BBSes follow a theme, whether that's old DOS-based adventures, ZX Spectrum fans, Commodore 64 gamers, or even something non-techie related, such as a Ford Cortina owners club; no doubt swapping owner manuals, old photos of the Cortina E and such.

In truth, a modern BBS is a bit of fun. Connecting to a system someone has installed and built, set around a particular theme, and designed with fantastic looking ANSI graphics, is a great pastime. It's a form of respect, in some ways, to acknowledge the work that's gone into creating the BBS by connecting to it and you also get to learn a little more about how protocols work, and how everything is connected.





HOW?

Connecting to a BBS via the Raspberry Pi is quite easy, but to get the most from it you will need to get your hands dirty in the Terminal.

As mentioned, we're going to be using a form of the protocol Telnet, in this instance specifically the program SyncTERM, to emulate the old-style Terminals that support ANSI art and IBM fonts, while connecting to the remote BBS with the Telnet protocol. You can simply use telnet under the Terminal (once you've installed it), but you'll miss some of the glorious artwork displayed within the majority of the BBSes.

To begin with, drop into a Terminal session on your Pi. When the Terminal is fired up, enter `sudo apt-get update` & `sudo apt-get upgrade` to ensure your system is up to date. If everything is okay, enter: `sudo apt-get install telnet`. While this stage isn't strictly necessary, it's always a good idea to have the base protocol client installed.

When telnet is installed, you can then start the procedure of installing SyncTERM. In order to get SyncTERM working, you'll need to build it from source. By now, you should be a dab hand at this, but here's the process in case you've forgotten (along with some added elements to help everything go to plan).

```
pi@raspberrypi: ~/Downloads
File Edit Tabs Help
pi@raspberrypi:~/Downloads $ ls
pi@raspberrypi:~/Downloads $ wget http://syncterm.bbsdev.net/syncterm-src.tgz
--2019-04-15 10:43:01-- http://syncterm.bbsdev.net/syncterm-src.tgz
Resolving syncterm.bbsdev.net (syncterm.bbsdev.net)... 2001:19f0:6000:94c9:
Connecting to syncterm.bbsdev.net (syncterm.bbsdev.net)|2001:19f0:6000:94c9:
HTTP request sent, awaiting response... 200 OK
Length: 851387 (8.1M) [application/octet-stream]
Saving to: 'syncterm-src.tgz'

syncterm-src.tgz      100%[=====]
2019-04-15 10:43:10 (971 KB/s) - 'syncterm-src.tgz' saved [851387/851387]

pi@raspberrypi:~/Downloads $ ls
syncterm-src.tgz
pi@raspberrypi:~/Downloads $
```

Begin by changing directory to the Downloads folder and downloading the source code:

```
cd Downloads\
wget http://syncterm.bbsdev.net/syncterm-src.tgz
ls
```

With `ls` entered, you should see the newly downloaded `tgz` file. To unpack the downloaded file, enter:

```
tar -xvf syncterm-src.tgz
```

This will create a new `syncterm-(DATE)` folder, where `DATE` is the current date when you've unpacked the contents of the `tgz` file.

You will now need to change directories to:

```
cd syncterm-(DATE)
cd src
cd syncterm
```

You can mesh these directories together, but, for the sake of keeping things simple, we'll stick to one folder at a time. Also, remember you can hit the `Tab` key to auto-complete a directory name.

```
pi@raspberrypi: ~/Downloads/syncterm-20190415
File Edit Tabs Help
pi@raspberrypi:~/Downloads $ ls
pi@raspberrypi:~/Downloads $ wget http://syncterm.bbsdev.net/syncterm-src.tgz
--2019-04-15 10:43:01-- http://syncterm.bbsdev.net/syncterm-src.tgz
Resolving syncterm.bbsdev.net (syncterm.bbsdev.net)... 2001:19f0:6000:94c9:
Connecting to syncterm.bbsdev.net (syncterm.bbsdev.net)|2001:19f0:6000:94c9:
HTTP request sent, awaiting response... 200 OK
Length: 851387 (8.1M) [application/octet-stream]
Saving to: 'syncterm-src.tgz'

syncterm-src.tgz      100%[=====]
2019-04-15 10:43:10 (971 KB/s) - 'syncterm-src.tgz' saved [851387/851387]

pi@raspberrypi:~/Downloads $ ls
syncterm-src.tgz
```

Once in the `syncterm` directory, you can begin to build from source. However, before you do that, it's worth installing a couple of extras to ensure the BBS session works to perfection. Start by installing the following:

```
sudo apt-get install libncurses5-dev
sudo apt-get install libsdl1.2-dev
```

Once these two are installed, start the build process by entering:

```
sudo make
```

Followed by:

```
sudo make install
```

```
pi@raspberrypi: ~/Downloads/syncterm-20190415
File Edit Tabs Help
cc -c -D_UNIX -DDEBUG -I. -DDATA_LITTLEENDIAN -DFIXED_SEED=0x17A85FAE
Ex -O2 -Wno-pointer-sign -Wno-strict-aliasing -fwrapv -fno-delete-null-pos
pr -D_REENTRANT test/utils.c
make[2]: Leaving directory '/home/pi/Downloads/syncterm-20190415/3rdp/src/'
make[1]: Leaving directory '/home/pi/Downloads/syncterm-20190415/3rdp/src/'
make[1]: Leaving directory '/home/pi/Downloads/syncterm-20190415/3rdp/build'
Compiling bbslist.c
Compiling uifcinit.c
Compiling ../uifc/filepick.c
Compiling fonts.c
Compiling rlogin.c
Compiling telnet_io.c
```

The process may take a few minutes, so be patient. When everything is installed, you can enter the command: `syncterm`, to start the program and change the screen settings, if you wish. However, to get straight into connecting to a BBS, try one of these commands:

```
syncterm dura-bbs.net: 6359
syncterm bbs.kernelerror.com: 10023
syncterm particlebbs.dyndns.org: 6400
syncterm heatwave.ddns.net: 9640
syncterm sysgod.org:23000
```

Naturally, some, or even all, of these BBSes may be offline when you come to test them; they are, after all, being operated by individuals like you and I. If they are offline, you can always get hold of a comprehensive list of active servers by visiting <https://www.telnetbbsguide.com/bbs/list/brief/>.

GET CONNECTED

It's worth spending some time finding the sort of BBS that suits your tastes. As you'll see by visiting the aforementioned website, there's over 500 BBSes currently listed, so somewhere in there a BBS could be your new online haunt.

Pi Projects: Weather Station

The Raspberry Pi has offered its userbase a superb platform to science. There are Raspberry Pi projects that involve the International Space Station, alongside NASA and ESA-led applications. However, it's an Earth-based project that's taken the community by storm.

IT'S RAINING PI, HALLELUJAH!

Weather stations have been available to the project-minded public since long before the Raspberry Pi entered the scene, but thanks to the Pi's unique specifications, a new generation of weather station has emerged; and they're really quite remarkable.

As with a lot of Raspberry Pi projects, the market has become flooded with various hardware, kits, and HATs that will enable you to set up a simple, or complex, weather station. The more experienced weather station and Raspberry Pi user will probably lean toward buying in the components individually, as opposed to an entire kit, but for the rest of us the kit-form versions are an excellent start to a project that can quickly grow over time.



WHY?

With our smart devices connected to everything around us all the time, having a multi-limbed weather station, perched precariously on the end of the garden shed, measuring the amount of rainfall and in what direction the wind is blowing, may seem a little unnecessary; since your phone could tell you all that already with just a swipe or two. However, that discounts the pleasure of creating your own project and measuring the results as they change from one day to the next.



Any project, regardless of whether it's on a Raspberry Pi or not, is something from which to learn. In the case of a Raspberry Pi driven weather station, the user can learn about all manner of electronics, connectivity, some simple engineering, and coding, even before we get onto the varying aspects of meteorology.

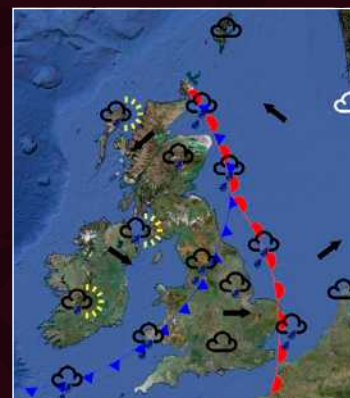
The weather station project doesn't necessarily need to be an all-powerful outdoor contraption either. Basically, there are two routes you can opt to take: the outdoor model, measuring wind speed/direction, precipitation and so on, and an indoor model



that measures ambient temperature, pressure and humidity.

Why indoors? There are a number of reasons. Perhaps you're in charge of a collection of servers, and while the company you work for may not be able to stretch to a fully sealed environment machine room, you could fill the gap with a great collection of sensors attached to a Raspberry Pi. You may have some form of allergy, where precise control of your indoor environment is important. Maybe you look after some exotic animals, in a room where air pressure, humidity and such are vital to the creature's health. Needless to say, there are plenty more examples.

In short, setting up a weather station is simply a fun project. Setting aside the things you'll learn from getting one up and running, you'll eventually have a physical piece of hardware that can, with moderate accuracy, measure your environment and display that data in a number of graphs and charts. It's quite cool, when you stop to think about it.





SOME OPTIONS?

To go into all the currently available Raspberry Pi weather station options will take more space than we're currently able to offer here, but let's have a quick look at some of the best selling kits and components.

Weather Meter Kit – For outdoor weather monitoring, we have the Weather Meter Kit as sold by CPC for around £77. The kit comprises of three main component sensors that measure wind speed, wind direction, and rainfall. The rain gauge is a self-emptying bucket that's capable of activating for each 0.011-inch of rainfall collected, while the anemometer activates a closure switch every 1.492mph once per second. The wind direction module can display up to sixteen different positions, and the kit comes complete with full assembly instructions.



Velleman WS3080 Weather Station – The Velleman WS3080 Weather Station is one of the higher-end products that can interact with the Raspberry Pi. This outdoor mounted station is capable of monitoring rainfall, wind speed and wind direction, air pressure, wind chill temperatures, air temperature, and UV light levels. It features alarms for temperature, humidity, wind chill, dew point, wind speed, air pressure and storms, and comes with a separate LCD transmitter.



Pimoroni Enviro Phat – The Enviro pHAT is an add-on sensor for the Raspberry Pi that includes four different sensors to monitor temperature, pressure, light levels, and motion. It's ideal for indoor environment monitoring and, thanks to some nifty Python libraries and code, you're able to display the data via a web page and view the variables remotely.



Cyntech WeatherHAT – The WeatherHAT from Cyntech can turn the Raspberry Pi into a weather display station. While not obtaining data directly from an external source (instead using one of the many localised, weather monitoring stations), it does offer seven RGB LEDs to highlight the current and upcoming weather. Controlled via Python, the HAT can help provide a good coding base for future weather station projects.

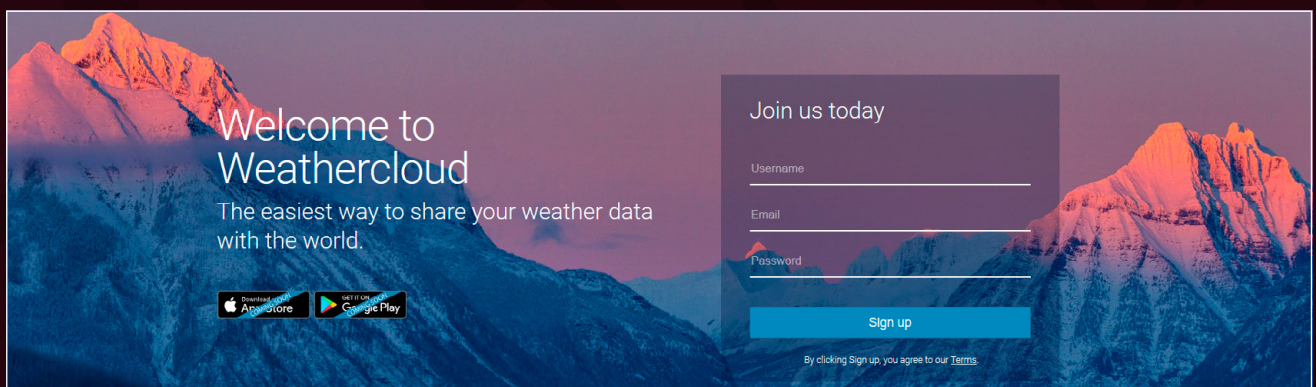


WeeWX – When you collect the necessary hardware together (the various sensors you'll need and so on), you'll need a good piece of software to help collate all that data and display it in a readable fashion; this is where WeeWX comes in. WeeWX is a Python-based program that can interact with many different weather stations to produce graphs and reports, and even publish the data to HTML on a web site.



WHERE NEXT?

As you can imagine, it'll take some time to get everything together, program the components, and collate the data you want to view. Once all that's gathered, though, what next?



There are some websites available that display localised weather from independent sources such as a back garden weather station, so when you've got everything up and running, try looking out for such projects and start collaborating. Also, consider creating a Twitter account for the Raspberry Pi weather station and Tweeting the local weather.

Stream Digital TV with Raspberry Pi

A HAT, in case you're wondering, is a Hardware Attached on Top add-on board that connects to a Raspberry Pi's 40-pin GPIO. They can extend the capabilities of a Pi by adding motors, LCDs, sensors, controllers and more. In addition, they can be programmed via the Pi and Python.

TV HAT

In this tutorial, we're using the Raspberry Pi TV HAT as sold by The Pi Hut (<https://thepihut.com/products/raspberry-pi-tv-hat?variant=13539182673982>). It's a Sony CXD2880 TV Tuner supporting DVB-T and 2nd-gen DVB-T2 standards.

STEP 1

The Raspberry Pi TV HAT is compatible with the Pi Zero, Pi 3 A+ and B+ models. The kit comes with everything you need to connect the HAT to your RPi, including spacers, screws and the 40-pin header. Begin by opening the box and spreading the contents out on a clear area.



STEP 2

Start by connecting the 40-pin header to the 40-pin GPIO on top of your Raspberry Pi. Now take the spacers and screws and fit them to the corresponding holes in the corners of the main Raspberry Pi and the TV HAT; use three spacers and screws for the Pi Zero and two for the Pi 3 Models A+/B+.



STEP 3

With the spacers attached to the Raspberry Pi, line up the TV HAT with the 40-pin header, ensuring that the side of the HAT with the Pi logo is facing up and the HAT's gold coloured coaxial attachment is on the side of the SD card. When in place, screw the HAT to its spacers.



STEP 4

Now attach the coaxial adapter to the gold-coloured pin at the side of the TV HAT. It may need a firm push to lock into place, it will 'click' when fully inserted and in position. Now you will need to connect the TV HAT to your TV aerial and provide power to the Raspberry Pi.



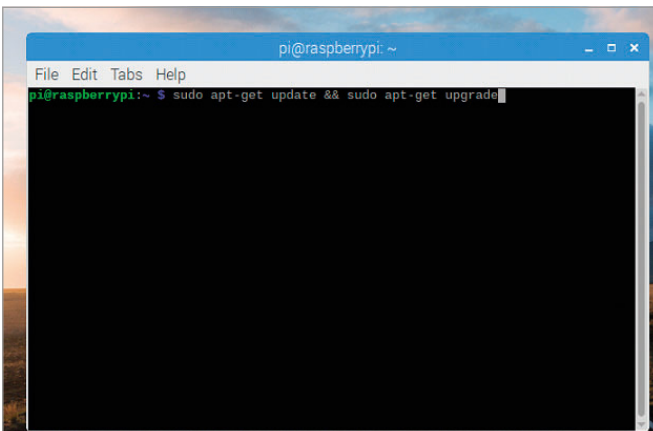
**STEP 5**

You can use the HDMI port to connect your Pi and its TV HAT to the TV/monitor, or you can just power the Pi (the TV HAT gets its power from the Pi) and connect remotely. The most important element is to ensure that the TV HAT is connected to an aerial that you know can receive a TV signal.

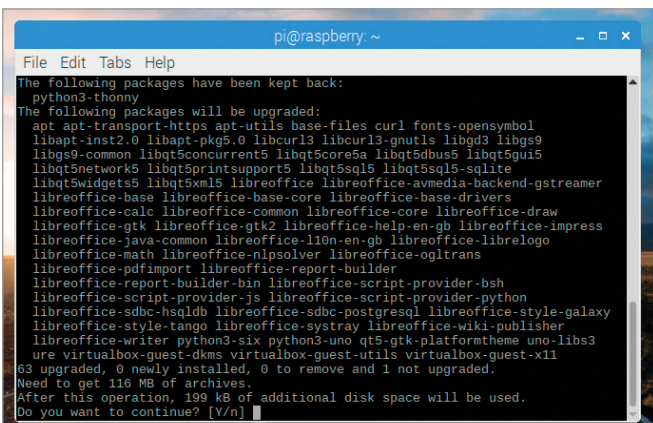
**STEP 6**

Power up your Raspberry Pi and when you get to the Raspbian desktop, open up the Terminal app. When at the Terminal, enter the following to ensure the Pi is up to date and running the latest versions of its software and system:

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

**STEP 7**

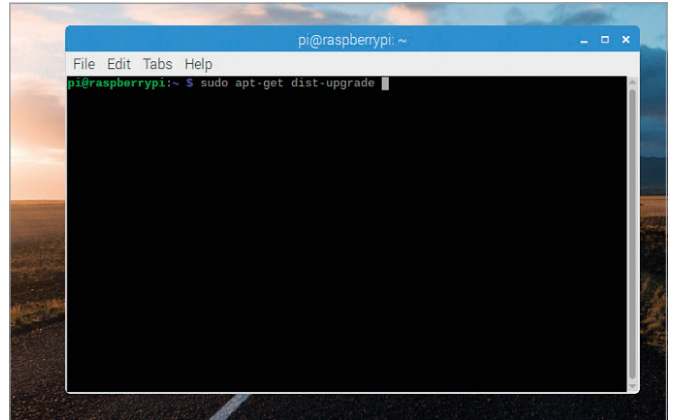
Press Enter and allow the Pi to run through the update process. If you have any significant updates, you may need to answer 'Y' to any questions the Pi asks regarding these. Answering yes will replace the older software with the newer versions and is necessary for a smoothly running Pi.

**STEP 8**

If you haven't used your Raspberry Pi since, at least, November 2018, then you may need to upgrade the core OS and synchronise the version of Raspbian with what's currently available from the Pi Foundation's downloads page. It's not totally necessary, but if you choose to, enter:

```
sudo apt-get dist-upgrade
```

Press Enter and follow the on-screen instructions.

**STEP 9**

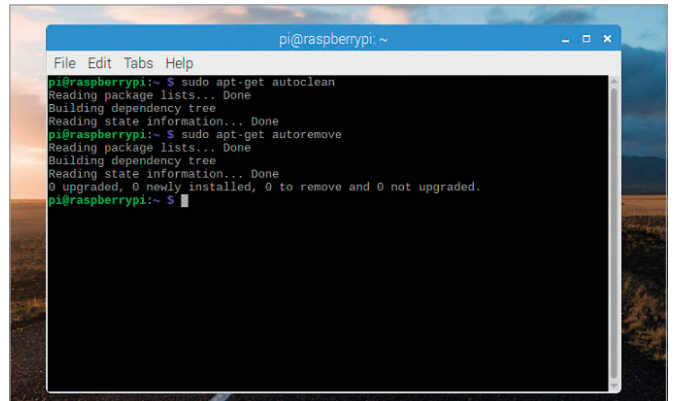
When the updates are complete, there's a good chance you've got some leftover, older setup files and packages in the system. To save space, use the following commands:

```
sudo apt-get autoclean
```

and:

```
sudo apt-get autoremove
```

Answer 'Y' to clear the system of the unnecessary packages and files.

**STEP 10**

At this point, it's worth noting that in the UK it's necessary to have a valid TV License. To use the TV tuner, we need to install the TVheadend software. Open a Terminal session and enter the following:

```
sudo apt-get install tvheadend
```

Press Enter and 'Y' if necessary to confirm the installation. Happy viewing!



Common Raspberry Pi Problems

The Raspberry Pi hardware and software is pretty reliable and problems are more often due to set up errors rather than anything to do with the hardware. However, there are times when hardware can seem to be at fault, so here are a few of the more common issues you might encounter when using your Raspberry Pi.

TROUBLESHOOTING YOUR RPI

The Raspberry Pi is a surprisingly stable bit of kit, but there is always the chance of encountering problems. If you're really stuck there are apps available which can help diagnose problems on the RPi.

RED POWER LED IS BLINKING

A blinking red power LED indicates problems with the power supply.

On model A and B, it is hard-wired to the 3.3V power supply rail. If it is blinking, it means the 5V power supply is dropping out. Use a different power supply. On the model B+ and also the A+, the circuit has been improved to give a much more reliable warning of poor power quality. The red power LED is wired to an APX803 supervisor which kicks in when the 5V power supply drops below 4.63V. If it does, the LED will blink. Check your connections, cable and power supply.

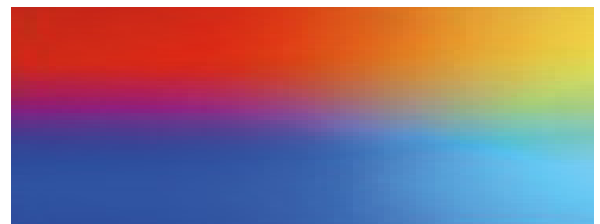


COLOURED SPLASH SCREEN

With the current firmware, a

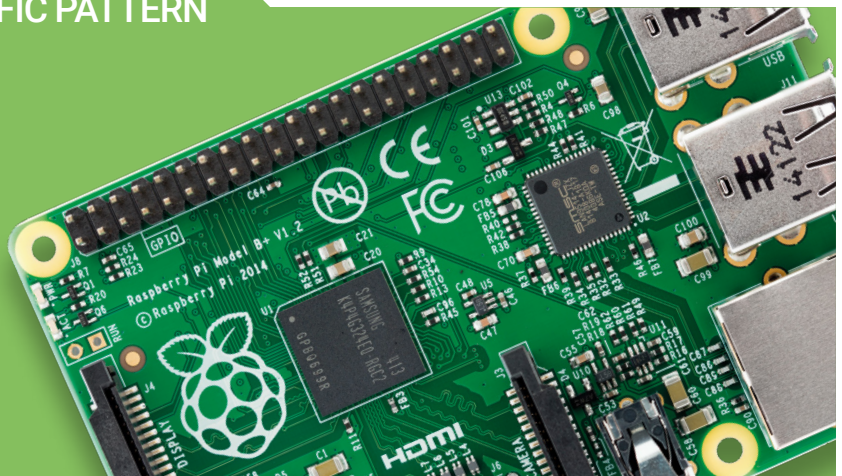
coloured splash screen is displayed after GPU firmware (start.elf) is loaded. This should be replaced by Linux console a second later. However if the coloured screen remains, it suggests the kernel.img file is failing to boot. Try replacing it with a known good one.

Immediately after displaying the splash screen, the Pi starts consuming a little more current. If the Pi resets at that moment, it is an indication that the power supply isn't able to deliver the full current that your Pi requires but dips its output voltage below a minimum when loaded with the full current the Pi needs.



GREEN LED BLINKS IN A SPECIFIC PATTERN

- 1 flash:** Possibly you have a RPi from Micron. Take a good look at the processor if it says M with an orbit round it. Using the latest software will solve your problem; also make sure you have a 4Gb SD card, as a 2Gb doesn't work.
- 2 flashes:** The SD Card cannot be read. A solution could be to format the card and flash Raspberry Pi OS with Pi Installer from Terminal.
- 3 flashes:** Start.elf not found.
- 4 flashes:** Start.elf not launched.
- 7 flashes:** Kernel.img not found.
- 8 flashes:** SDRAM not recognised. You need newer bootcode.bin/start.elf firmware.





NO USB DEVICE WORKS

The most common cause of USB devices working is low power supply voltage from a bad PSU, cable or USB hub; but it could also be that no clock signal is present. Return the board for a replacement if you think this is the case but before coming to this conclusion, confirm known good peripherals. A significant number of USB keyboards are not compatible with Raspberry so make sure you are using one made for Pi.



RASPBERRY PI NOT RESPONDING TO KEY PRESSES

This is most often caused by inadequate power. Use a good power supply and a good power cable. Some cheap cables that work with a mobile phone, cannot fully power the Pi. Some USB devices require a lot of power; most will have a label showing the voltage and mA requirements. Each one should be 5v 100mA max. Any more than this and they must be used with a powered USB hub. Try unplugging every USB device except the keyboard. You should also note that some keyboards have built in hubs and can try to draw 150mA; Pi can only handle 100mA per USB slot without a hub. Use the latest software too.



KEYBOARD OR MOUSE INTERFERES WITH A USB WI-FI DEVICE

Connecting a keyboard and or

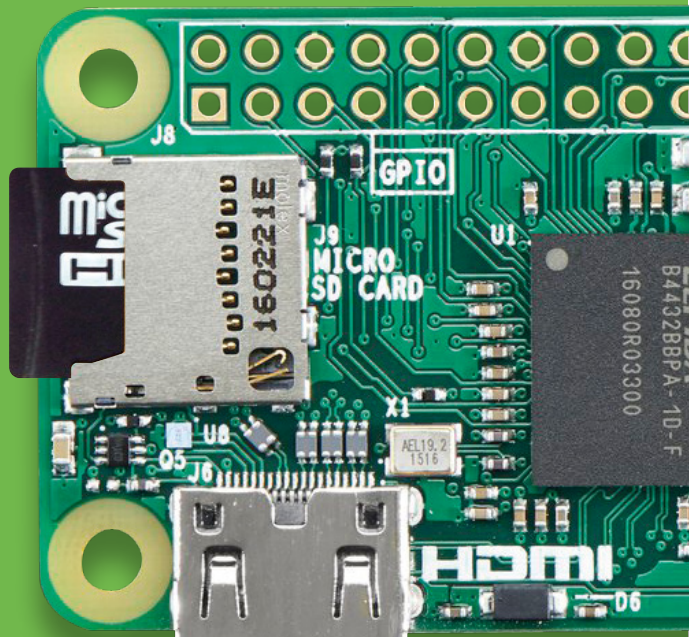
mouse while a USB Wi-Fi device is connected, may cause one or both devices to malfunction. Tests point to interferences in the 2.4 GHz frequency band in which both Wi-Fi sticks, as well as USB keyboards transmit data. Changing the channel on the wireless access point should fix the problem completely.



SD CARD PROBLEMS

If you have problems, check you have the latest firmware version first. If that is not the problem, try the following.

- Some SD cards do not work on the Pi, so check the list of known SD cards on the official Pi website.
- If you are having problems setting up your SD card you might want to start by erasing it completely, especially if it has been used elsewhere and still contains data or partitions.
- Windows and Mac users can download a formatting tool from the SD Association: https://www.sdcard.org/downloads/formatter_3/
- Reformatting cards is also easy to do in a digital camera.
- After writing the image to the SD card, verify that you can see the boot partition when you insert the SD card into your computer. The partition should contain a number of files, including start.elf and kernel.img. If you do not see these files on the SD card, you have made an error writing the image file.
- If you are manually preparing your SD card on Linux or macOS using the dd command, this operation will completely erase any existing data and partitions. Make sure you write to the whole card (e.g. /dev/sdd) and not to an existing partition (e.g. /dev/sdd1).
- If you put the SD card into your PC in an attempt to write the Pi operating system onto it and the PC tells you the card is write-protected, even with the write-protect tab in the correct forward position you may have a faulty SD-card rewiter.



Save a Huge 50% Off ALL Digital Manuals!*



As a token of our thanks for your support,  **Papercut** bring you this unmissable offer! Not only can you learn new skills and master your tech, but you can now **SAVE 50%** off all of our coding, photography and consumer tech digital manuals!

To take advantage of this amazing offer visit:

www.pclpublications.com/product-category/digital/

Make your selection(s) and then simply use the following exclusive code at checkout to get your purchase at half price:

SAVE50

*Please note this exclusive single use per user discount code requires sign up for activation. Discount cannot be applied against discounted items, print titles or used with an additional discount code.

Want to master your Code?

Then don't miss our **NEW** Programming magazine on  Readly now!



Raspberry Pi Tricks & Tips

23 | ISBN: 978-1-912847-70-9

Published by: Papercut Limited
Digital distribution by: Readly AB

©2026 Papercut Ltd. 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 contents for whatever purpose. Any app, hardware or software images reproduced on the front cover are solely for design purposes and are not necessarily representative of content. We advise all potential buyers to check listings prior to purchase for confirmation of actual content.

All editorial 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 or content.

This is an independent publication and as such does not necessarily reflect the views or opinions of the producers of apps, software or products contained within. This publication is 100% unofficial and in no way associated with any other company, app developer, software developer or manufacturer. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery

reproduced with courtesy of brands, apps, software and product manufacturers. Additional images are reproduced under licence from Shutterstock. Prices, international availability, ratings, titles and content are subject to change. Some content may have been previously published in other editions. All information was correct at time of publication.



Papercut Limited
Registered in England & Wales No: 04308513

INTERNATIONAL LICENSING - Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: jgale@pctlpublications.com