TRICKS & TIPS

Python Coding

Everything you need to take your Python programming skills to the next level

- Advanced guides and tutorials for programming C++ and Python

OVER 445 SECRETS & HACKS

- Next level fixes and secrets get to the heart of the Python language
- Unlock the power of code with games, graphics, animation and much more

Get Your Exclusive FREE Gift Worth £9.99 Here!

Download Your FREE Copy of Tech Shopper Magazine

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

1/ Enter the following URL: **www.pclpublications.com/exclusives**
2/ Sign up/in and from the listings of our exclusive customer downloads, highlight the Tech Shopper Magazine option.
3/ Enter your unique download code (Listed below) in the "Enter download code" bar.
4/ Click the Download Now! Button and your file will automatically download.
5/ Your file is a high resolution PDF file, which is compatible with the majority of customer devices/platforms.

Exclusive Download Code: **PCL37862RE**

NOTE: This is a limited reader offer running from 1st July 2023 to 1st December 2023.
Terms and conditions apply and can be found at www.pclpublications.com

Python Coding

Python Coding 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 coding and ultimately Python coding itself. 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.

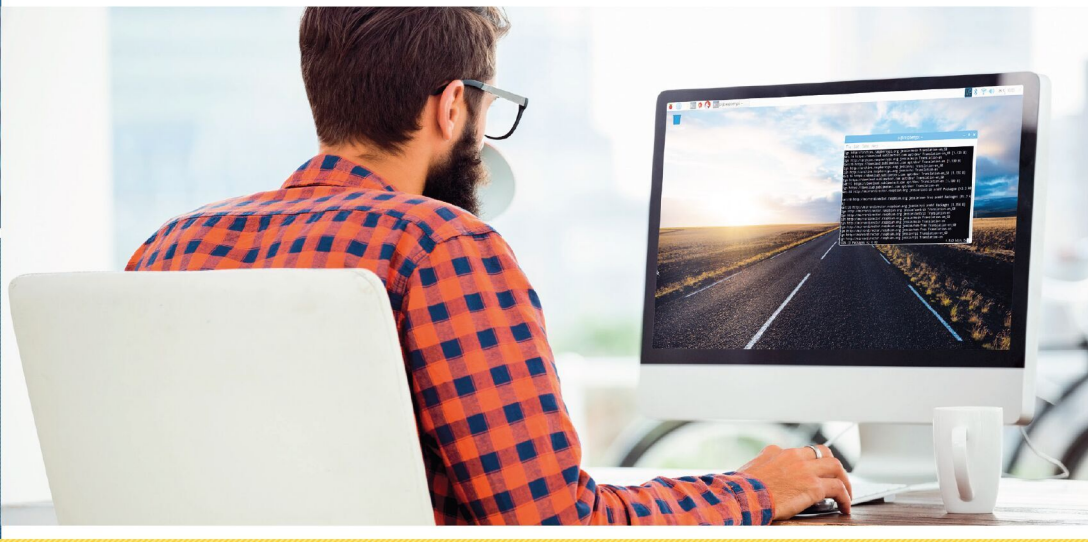
Over the page our journey continues, and we will be with you at every stage to advise, inform and ultimately inspire you to go further.

Contents

Using Modules

Code Repository

Understanding Linux



"There's so much you can do with Python and within these pages you'll find everything you need to know to become a Python programmer, ready for the next level of advanced coding."


mirror object to mirror
mirror_mod.mirror_object
operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y"
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z"
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end -add
ob.select= 1
er_ob.select=1
ntext.scene.objects.acti
Selected" + str(modifi
irror_ob.select = 0
bpy.context.selected_ob
ata.objects[one.name].se
int("please select exactly
OPERATOR CLASSES

Using Modules

A Python module is a Python-created source file that contains the necessary code for classes, functions and global variables. You can bind and reference modules to extend functionality, and create even more spectacular Python programs.

Are you curious about how to improve your use of these modules to add a little something extra to your code? Then read on and learn how they can be used to fashion fantastic code with graphics, animations and operating system specific commands.

Calendar Module

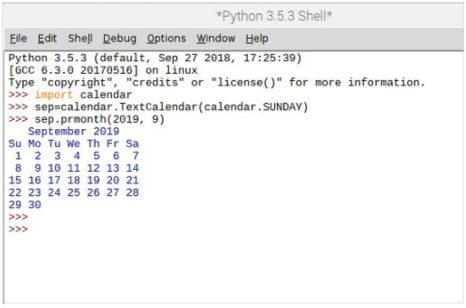
Beyond the Time module, the Calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the Time module-like format, you can actually call up a wall calendar type display.

WORKING WITH DATES

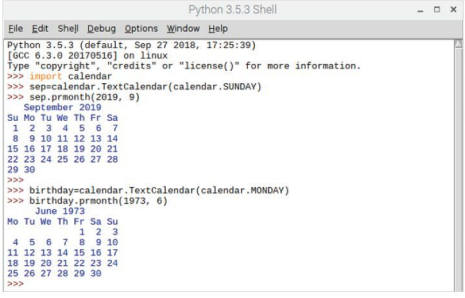
The Calendar module is built into Python 3. However, if for some reason it's not installed you can add it using pip install calendar as a Windows administrator, or sudo pip install calendar for Linux and macOS.

STEP 1 Launch Python 3 and enter: `import calendar` to call up the module and its inherent functions. Once it's loaded into memory, start by entering:

```
sep=calendar.TextCalendar(calendar.SUNDAY)
sep.prmonth(2019, 9)
```



STEP 2 You can see that the days of September 2019 are displayed in a wall calendar fashion. Naturally you can change the 2019, 9 part of the second line to any year and month you want, a birthday for example (1973, 6). The first line configures TextCalendar to start its weeks on a Sunday; you can opt for Monday if you prefer.



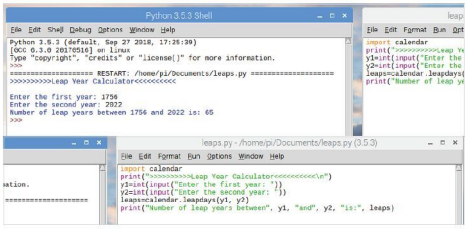
STEP 3 There are numerous functions within the Calendar module that may be of interest to you when forming your own code. For example, you can display the number of leap years between two specific years:

```
leaps=calendar.leapdays(1900, 2019)
print(leaps)
```

The result is 29, starting from 1904 onward.

STEP 4 You could even fashion that particular example into a piece of working, user interactive Python code:

```
import calendar
print(">>>>>>>>>>Leap Year Calculator<<<<<<<<<<\n")
y1=int(input("Enter the first year: "))
y2=int(input("Enter the second year: "))
leaps=calendar.leapdays(y1, y2)
print("Number of leap years between", y1, "and",
y2, "is:", leaps)
```

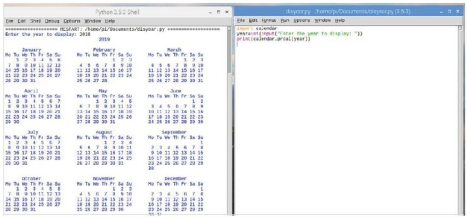


STEP 5

You can also create a program that will display all the days, weeks and months within a given year:

```
import calendar
year=int(input("Enter the year to display: "))
print(calendar.prcal(year))
```

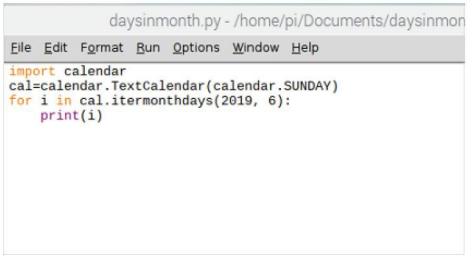
We're sure you'll agree that's quite a handy bit of code to have to hand.



STEP 6

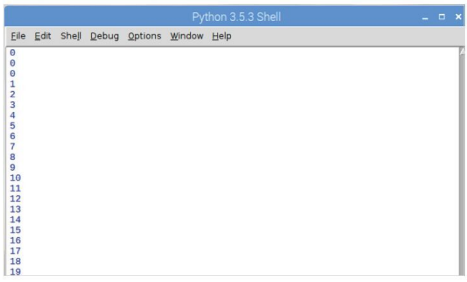
Interestingly we can also list the number of days in a month by using a simple: for loop:

```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2019, 6):
        print(i)
```



STEP 7

You can see that, at the outset, the code produced some zeros. This is due to the starting day of the week, Sunday in this case, plus overlapping days from the previous month. Meaning the counting of the days will start on Saturday 1st June 2019 and will total 30, as the output correctly displays.

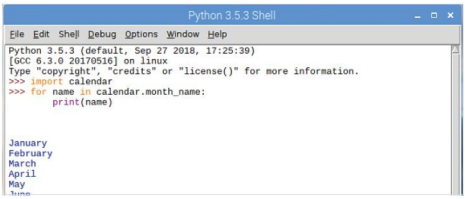


STEP 8

You're also able to print the individual months, or days, of the week:

```
import calendar
for name in calendar.month_name:
   print(name)
```

```
import calendar
for name in calendar.day_name:
   print(name)
```

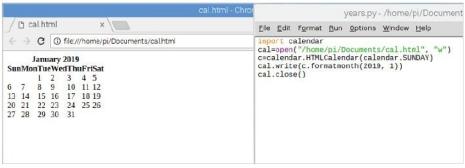


STEP 9

The Calendar module also allows us to write the functions in HTML, so that you can display it on a website. Let's start by creating a new file:

```
import calendar
cal=open("/home/pi/Documents/cal.html", "w")
c=calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2019, 1))
cal.close()
```

This code will create an HTML file called cal, open it with a browser and it displays the calendar for January 2019.

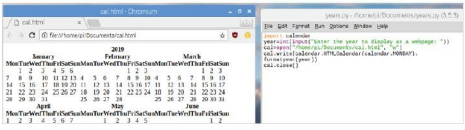


STEP 10

Of course, you can modify that to display a given year as a web page calendar:

```
import calendar

year=int(input("Enter the year to display as a
webpage: "))
cal=open("/home/pi/Documents/cal.html", "w")
cal.write(calendar.HTMLCalendar(calendar.MONDAY).
formatyear(year))
cal.close()
```

This code asks the user for a year and then creates the necessary webpage. Remember to change your file destination.



OS Module

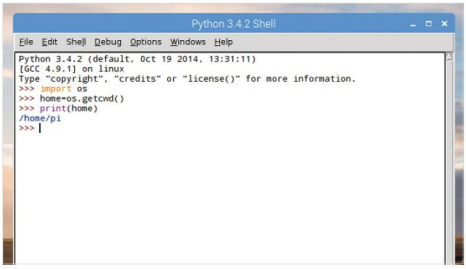
The OS module allows you to interact directly with the built-in commands found in your operating system. Commands vary depending on the OS you're running, as some will work with Windows whereas others will work with Linux and macOS.

INTO THE SYSTEM

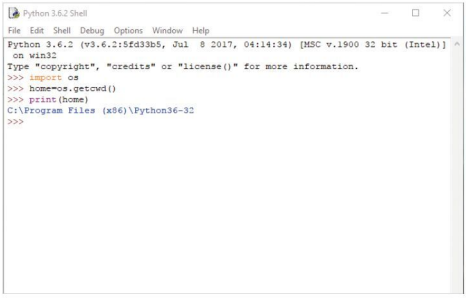
One of the primary features of the OS module is the ability to list, move, create, delete and otherwise interact with files stored on the system, making it the perfect module for backup code.

STEP 1 You can start the OS module with some simple functions to see how it interacts with the operating system environment that Python is running on. If you're using Linux or the Raspberry Pi, try this:

```
import os
home=os.getcwd()
print(home)
```



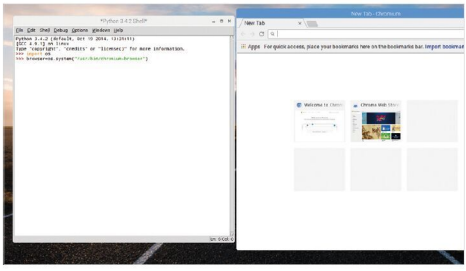
STEP 2 The returned result from printing the variable home is the current user's home folder on the system. In our example that's /home/pi; it will be different depending on the user name you log in as and the operating system you use. For example, Windows 10 will output: C:\Program Files (x86)\Python36-32.



STEP 3 The Windows output is different as that's the current working directory of Python, as determined by the system; as you might suspect, the os.getcwd() function is asking Python to retrieve the Current Working Directory. Linux users will see something along the same lines as the Raspberry Pi, as will macOS users.

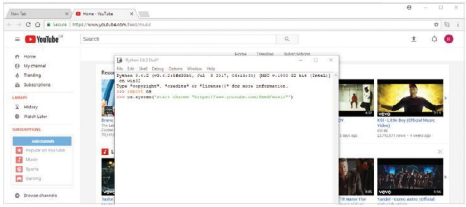
STEP 4 Yet another interesting element to the OS module, is its ability to launch programs that are installed in the host system. For instance, if you wanted to launch the Chromium browser from within a Python program you can use the command:

```
import os
browser=os.system("/usr/bin/chromium-browser")
```



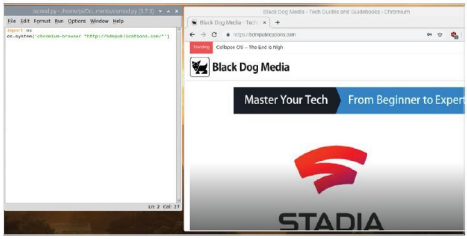
STEP 5 The os.system() function is what allows interaction with external programs; you can even call up previous Python programs using this method. You will obviously need to know the full path and program file name for it to work successfully. However, you can use the following:

```
import os
os.system('start chrome "https://www.youtube.com/
feed/music"')
```



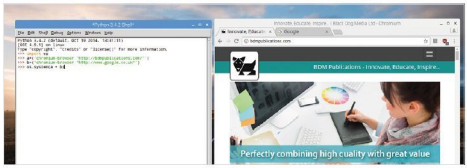
STEP 6 For Step 5's example we used Windows, to show that the OS module works roughly the same across all platforms. In that case, we opened YouTube's music feed page, so it is therefore possible to open specific pages:

```
import os
os.system('chromium-browser "http://
bdmpublications.com/"')
```



STEP 7 Note in the previous step's example the use of single and double-quotes. The single quotes encase the entire command and launching Chromium, whereas the double quotes open the specified page. You can even use variables to call multiple tabs in the same browser:

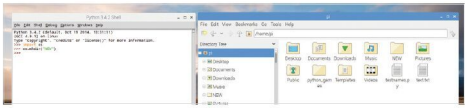
```
import os
a=('chromium-browser "http://bdmpublications.
com/"')
b=('chromium-browser "http://www.google.co.uk"')
os.system(a + b)
```



STEP 8 The ability to manipulate directories, or folders if you prefer, is one of the OS module's best features. For example, to create a new directory you can use:

```
import os
os.mkdir("NEW")
```

This creates a new directory within the Current Working Directory, named according to the object in the mkdir function.

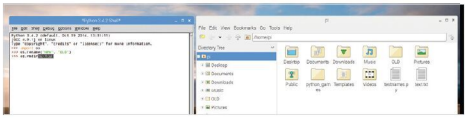


STEP 9 You can also rename any directories you've created by entering:

```
import os
os.rename("NEW", "OLD")
```

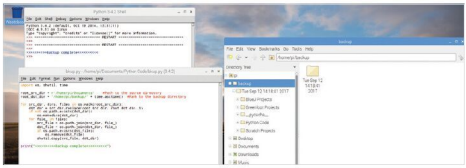
To delete them:

```
import os
os.rmdir("OLD")
```



STEP 10 Another module that goes together with OS is shutil. You can use the Shutil module together with OS and time to create a time-stamped backup directory, and copy files into it:

```
import os, shutil, time

root_src_dir = r'/home/pi/Documents'
root_dst_dir = '/home/pi/backup/' + time.asctime()

for src_dir, dirs, files in os.walk(root_src_dir):
  dst_dir = src_dir.replace(root_src_dir, root_
dst_dir, 1)
  if not os.path.exists(dst_dir):
    os.makedirs(dst_dir)
  for file_ in files:
    src_file = os.path.join(src_dir, file_)
    dst_file = os.path.join(dst_dir, file_)
    if os.path.exists(dst_file):
      os.remove(dst_file)
    shutil.copy(src_file, dst_dir)

print(">>>>>>>>>>Backup complete<<<<<<<<<<")
```



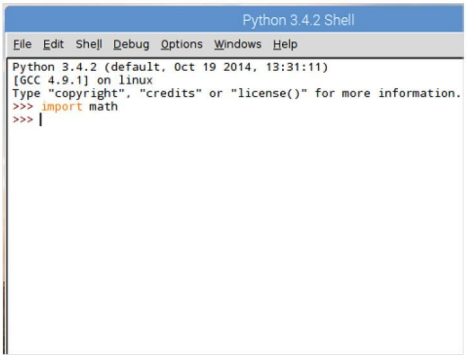
Using the Math Module

One of the most used modules you will come across is the Math module. As we've mentioned previously in this book, mathematics is the backbone of programming and there's an incredible number of uses the Math module can have in your code.

$E = MC^2$

The Math module provides access to a plethora of mathematical functions, from simply displaying the value of Pi, to helping you create complex 3D shapes.

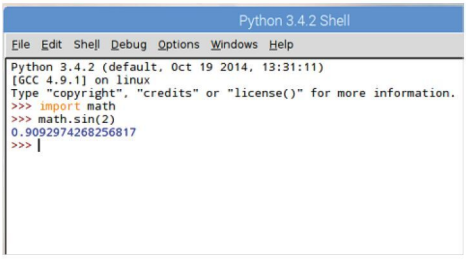
STEP 1 The Math module is built-in to Python 3; so there's no need to PIP install it. As with the other modules present, you can import the module's function by simply entering `import math` into the Shell, or as part of your code in the Editor.



STEP 2 Importing the Math module will give you access to the module's code. From there, you can call up any of the available functions within Math by using `math`, followed by the name of the function in question. For example, enter:

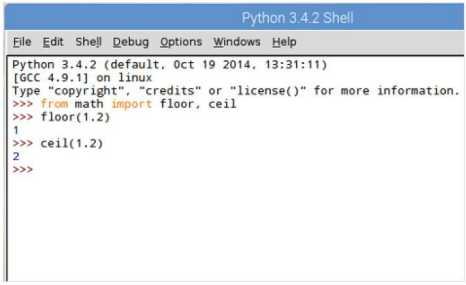
```
math.sin(2)
```

This displays the sine of 2.



STEP 3 As you will no doubt be aware by now, if you know the name of the individual functions within the module you can specifically import them. For instance, the Floor and Ceil functions round down and up a float:

```
from math import floor, ceil
floor(1.2) # returns 1
ceil(1.2) # returns 2
```



STEP 4 The Math module can also be renamed as you import it, as with the other modules on offer within Python. This often saves time, but don't forget to make a comment to show someone else looking at your code what you've done:

```
import math as m
m.trunc(123.45) # Truncate removes the fraction
```

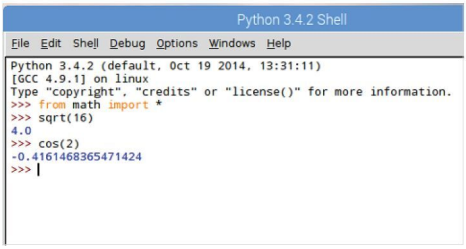
STEP 5 Although it's not common practise, it is possible to import functions from a module and rename them. In this example, we're importing Floor from Math and renaming it to f. Although where lengthy code is in use, this process can quickly become confusing:

```
from math import floor as f
f(1.2)
```

STEP 6 Importing all the functions of the Math Module can be done by entering:

```
from math import *
```

While certainly handy, this is often frowned upon by the developer community as it takes up unnecessary resources and isn't an efficient way of coding. However, if it works for you then go ahead.



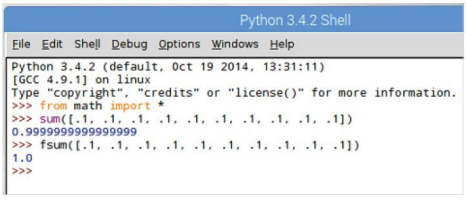
STEP 7 Interestingly, some functions within the Math module are more accurate, or to be more precise are designed to return a more accurate value, than others. For example:

```
sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
```

will return the value of 0.999999999. Whereas:

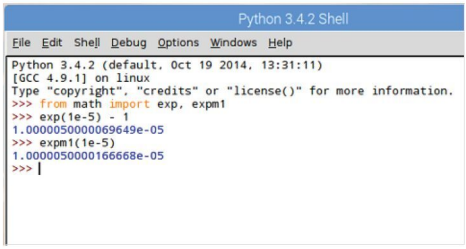
```
fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
```

returns the value of 1.0.



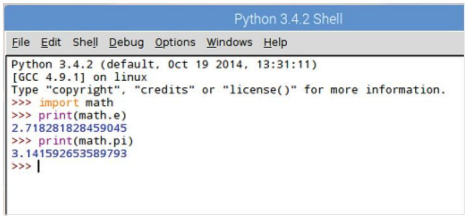
STEP 8 For further accuracy, when it comes to numbers the exp and expm1 functions can be used to compute precise values:

```
from math import exp, expm1
exp(1e-5) - 1 # value accurate to 11 places
expm1(1e-5) # result accurate to full precision
```

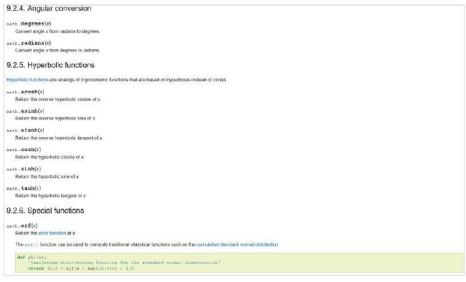


STEP 9 This level of accuracy is really quite impressive, but quite niche for the most part. Probably the two most used functions are **e** and **Pi**, where e is the numerical constant equal to 2.71828 (where the circumference of a circle is divided by its diameter):

```
import math
print(math.e)
print(math.pi)
```



STEP 10 The wealth of mathematical functions available through the Math module is vast and covers everything from factors to infinity, powers to trigonometry and angular conversion to constants. Look up **https://docs.python.org/3/library/math.html#** for a list of available Math module functions.


9.2.4. Angular conversion

math.degrees(x)
Convert angle x from radians to degrees.

math.radians(x)
Convert angle x from degrees to radians.

9.2.5. Hyperbolic functions

Hyperbolic functions are analogs or trigonometric functions that are based on hyperbolas instead or circles.

math.acosh(x)
Return the inverse hyperbolic cosine of x.

math.asinh(x)
Return the inverse hyperbolic sine of x.

math.atanh(x)
Return the inverse hyperbolic tangent of x.

math.cosh(x)
Return the hyperbolic cosine of x.

math.sinh(x)
Return the hyperbolic sine of x.

math.tanh(x)
Return the hyperbolic tangent of x.

9.2.6. Special functions

math.erf(x)
Return the error function at x.

The erf() function can be used to compute traditional statistical functions such as the cumulative standard normal distribution:

```
def phi(x):
    'Cumulative distribution function for the standard normal distribution'
    return (1.0 + erf(x / sqrt(2.0))) / 2.0
```

Random Module

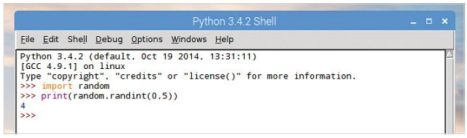
The Random module is one you will likely come across many times in your Python programming lifetime; as the name suggests, it's designed to create random numbers or letters. However, it's not exactly random but it will suffice for most needs.

RANDOM NUMBERS

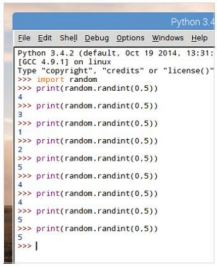
There are numerous functions within the Random module, which when applied can create some interesting and very useful Python programs.

STEP 1 Just as with other modules you need to import random before you can use any of the functions we're going to look at in this tutorial. Let's begin by simply printing a random number from 1 to 5:

```
import random
print(randomint(0,5))
```



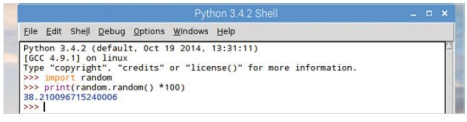
STEP 2 In our example the number four was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.



STEP 3 For a bigger set of numbers, including floating point values, you can extend the range by using the multiplication sign:

```
import random
print(random.random() *100)
```

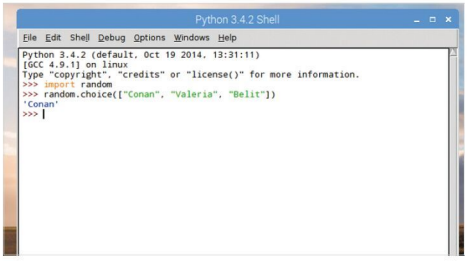
Will display a floating point number between 0 and 100, to the tune of around fifteen decimal points.



STEP 4 However, the Random module isn't used exclusively for numbers. You can use it to select an entry from a list from random, and the list can contain anything:

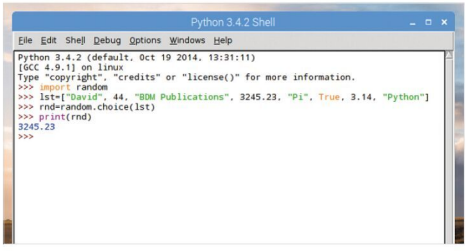
```
import random
random.choice(["Conan", "Valeria", "Belit"])
```

This will display one of the names of our adventurers at random, which is a great addition to a text adventure game.



STEP 5 You can extend the previous example somewhat by having random.choice() select from a list of mixed variables. For instance:

```
import random
lst=["David", 44, "BDM Publications", 3245.23,
"Pi", True, 3.14, "Python"]
rnd=random.choice(lst)
print(rnd)
```

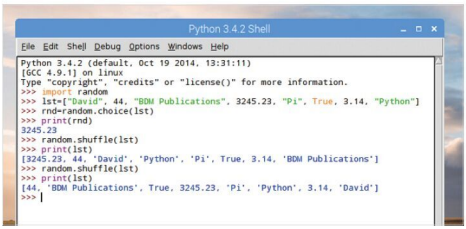


STEP 6

Interestingly, you can also use a function within the Random module to shuffle the items in the list, thus adding a little more randomness into the equation:

```
random.shuffle(lst)
print(lst)
```

This way, you can keep shuffling the list before displaying a random item from it.

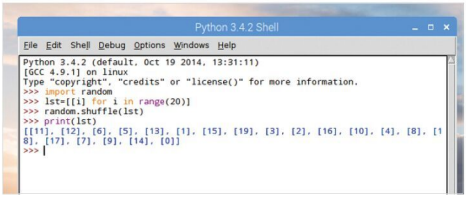


STEP 7

Using shuffle, you can create an entirely random list of numbers. For example, within a given range:

```
import random
lst=[[i] for I in range(20)]
random.shuffle(lst)
print(lst)
```

Keep shuffling the list and you can have a different selection of items from 0 to 20 every time.

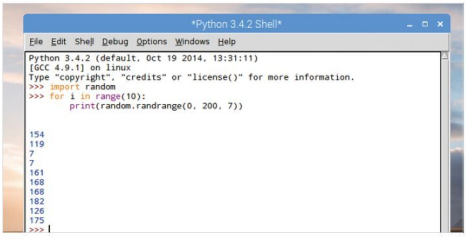


STEP 8

You can also select a random number from a given range in steps, using the start, stop, step loop:

```
import random
for i in range(10):
  print(random.randrange(0, 200, 7))
```

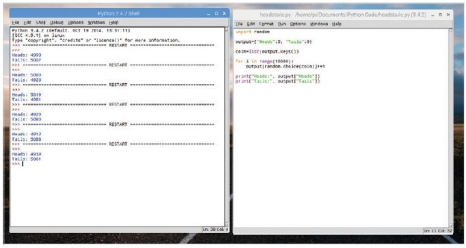
Results will vary but you get the general idea as to how it works.



STEP 9

Let's use an example piece of code which flips a virtual coin ten thousand times and counts how many times it will land on heads or tails:

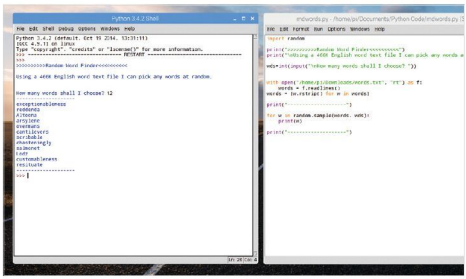
```
import random
output={"Heads":0, "Tails":0}
coin=list(output.keys())

for i in range(10000):
  output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```



STEP 10

Here's an interesting piece of code. Using a text file containing 466 thousand words, you can pluck a user generated number of words from the file (text file found at: www.github.com/dwyl/english-words):

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can pick any words at random.\n")

wds=int(input("\nHow many words shall I choose? "))

with open("/home/pi/Downloads/words.txt", "rt") as f:
  words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
  print(w)

print("--------------------")
```



Tkinter Module

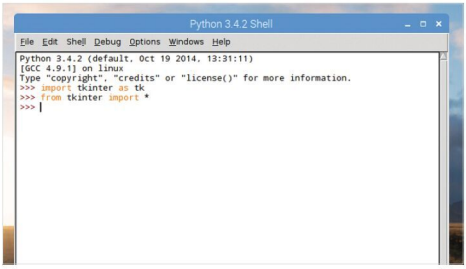
While running your code from the command line, or even in the Shell, is perfectly fine, Python is capable of so much more. The Tkinter module enables the programmer to set up a Graphical User Interface to interact with the user, and it's surprisingly powerful too.

GETTING GUI

Tkinter is easy to use but there's a lot more you can do with it. Let's start by seeing how it works and getting some code into it. Before long you will discover just how powerful this module really is.

STEP 1 Tkinter is usually built into Python 3. However, if it's available when you enter: `import tkinter`, then you need to `pip install tkinter` from the command prompt. We can start to import modules differently than before, to save on typing and by importing all their contents:

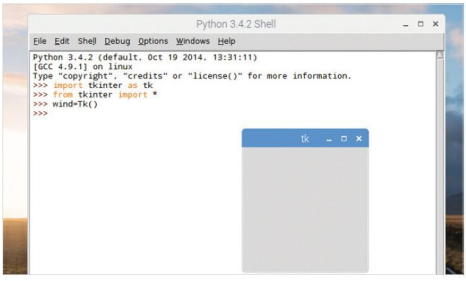
```
import tkinter as tk
from tkinter import *
```



STEP 2 It's not recommended to import everything from a module using the asterisk but it won't do any harm normally. Let's begin by creating a basic GUI window, enter:

```
wind=Tk()
```

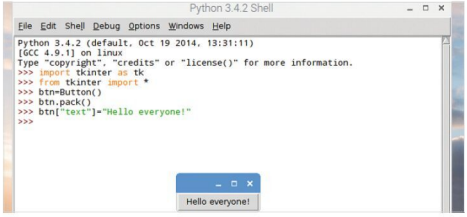
This creates a small, basic window. There's not much else to do at this point but click the X in the corner to close the window.



STEP 3 The ideal approach is to add mainloop() into the code to control the Tkinter event loop, but we'll get to that soon. You've just created a Tkinter widget and there are several more we can play around with:

```
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

The first line focuses on the newly created window. Click back into the Shell and continue the other lines.

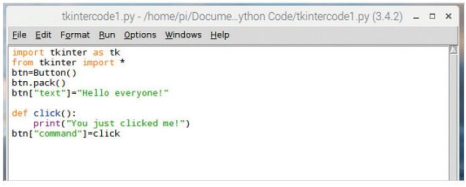


STEP 4 You can combine the above into a New File:

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"
```

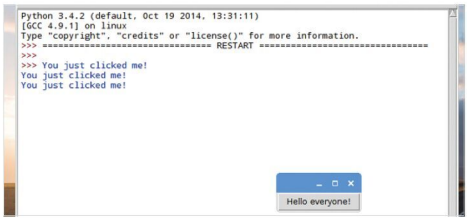
Then add some button interactions:

```
def click():
  print("You just clicked me!")
btn["command"]=click
```



STEP 5

Save and execute the code from Step 5 and a window appears with 'Hello everyone!' inside. If you click the Hello everyone! button, the Shell will output the text 'You just clicked me!'. It's simple but shows you what can be achieved with a few lines of code.



STEP 6

You can also display both text and images within a Tkinter window. However, only GIF, PGM or PPM formats are supported. So find an image and convert it before using the code. Here's an example using the BDM Publishing logo:

```
from tkinter import *

root = Tk()
logo = PhotoImage(file="/home/pi/Downloads/BDM_logo.
gif")
w1 = Label(root, root.title("BDM Publications"),
image=logo).pack(side="right")
content = """  From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication
produced by a team of just two to one of the biggest
names in global bookazine publishing, for two simple
reasons. Our passion and commitment to deliver the
very best product each and every volume. While
the company has grown with a portfolio of over 250
publications delivered by our international staff,
the foundation that it has been built upon remains
the same, which is why we believe BDM isn't just
the first choice it's the only choice for the smart
consumer.  """
w2 = Label(root,
    justify=LEFT,
    padx = 10,
    text=content).pack(side="left")
root.mainloop()
```

STEP 7

The previous code is quite weighty, mostly due to the content variable holding a part of BDM's About page from the company website. You can obviously change the content, the root.title and the image to suit your needs.

STEP 8

You can create radio buttons too. Try:

```
from tkinter import *

root = Tk()

v = IntVar()

Label(root, root.title("Options"), text="""Choose
a preferred language:""",
  justify = LEFT, padx = 20).pack()
Radiobutton(root,
    text="Python",
    padx = 20,
    variable=v,
    value=1).pack(anchor=W)
Radiobutton(root,
    text="C++",
    padx = 20,
    variable=v,
    value=2).pack(anchor=W)

mainloop()
```

STEP 9

You can also create check boxes, with buttons and output to the Shell:

```
from tkinter import *
root = Tk()

def var_states():
  print("Warrior: %d,\nMage: %d" % (var1.get(),
var2.get()))

Label(root, root.title("Adventure Game"),
text=">>>>>>>>>>Your adventure role<<<<<<<<<<").
grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text="Warrior", variable=var1).
grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text="Mage", variable=var2).
grid(row=2, sticky=W)
Button(root, text='Quit', command=root.destroy).
grid(row=3, sticky=W, pady=4)
Button(root, text='Show', command=var_states).
grid(row=3, sticky=E, pady=4)

mainloop()
```

STEP 10

The code from Step 9 introduced some new geometry elements into Tkinter. Note the sticky=N, E and W arguments. These describe the locations of the check boxes and buttons (North, East, South and West). The row argument places them on separate rows. Have a play around and see what you get.

Pygame Module

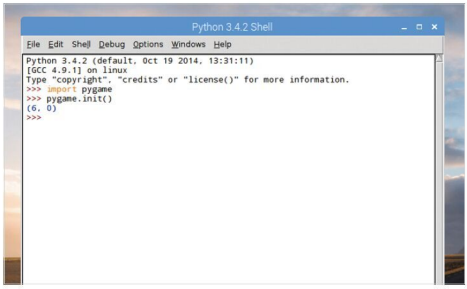
We've had a brief look at the Pygame module already but there's a lot more to it that needs exploring. Pygame was developed to help Python programmers create either graphical or text-based games.

PYGAMING

Pygame isn't an inherent module to Python but those using the Raspberry Pi will already have it installed. Everyone else will need to use: pip install pygame from the command prompt.

STEP 1 Naturally you need to load up the Pygame modules into memory before you're able to utilise them. Once that's done Pygame requires the user to initialise it prior to any of the functions being used:

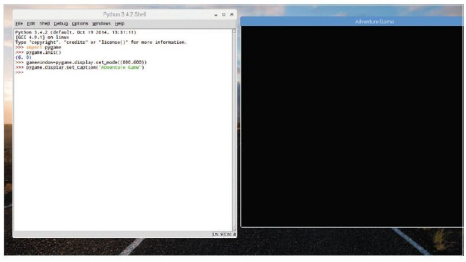
```
import pygame
pygame.init()
```



STEP 2 Let's create a simple game ready window, and give it a title:

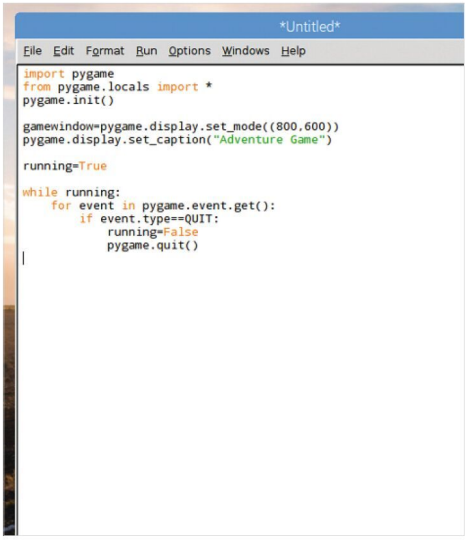
```
gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
```

You can see that after the first line is entered, you need to click back into the IDLE Shell to continue entering code; also, you can change the title of the window to anything you like.



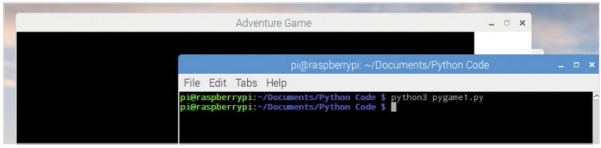
STEP 3 Sadly you can't close the newly created Pygame window without closing the Python IDLE Shell, which isn't very practical. For this reason, you need to work in the editor (New > File) and create a True/False while loop:

```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")

running=True

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



STEP 4

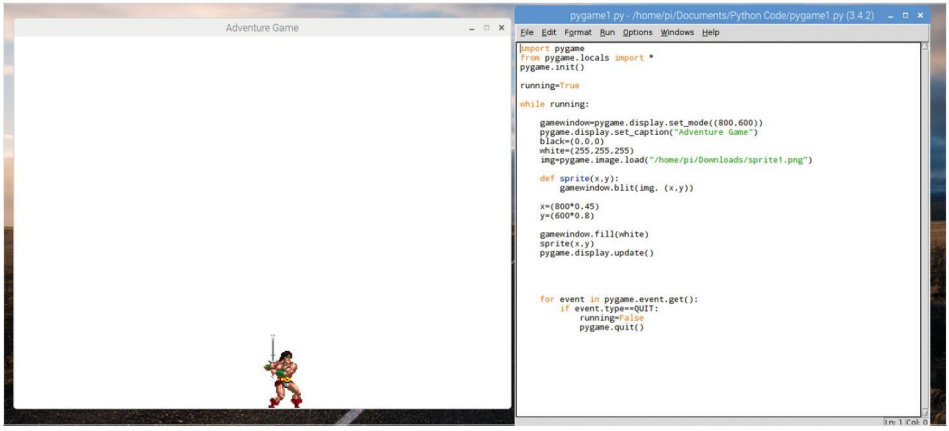
If the Pygame window still won't close don't worry, it's just a discrepancy between the IDLE (which is written with Tkinter) and the Pygame module. If you run your code via the command line, it closes perfectly well.



STEP 5

You're going to shift the code around a bit now, running the main Pygame code within a while loop; it makes it neater and easier to follow. We've downloaded a graphic to use and we need to set some parameters for pygame:

```
import pygame
pygame.init()

running=True

while running:

  gamewindow=pygame.display.set_mode((800,600))
  pygame.display.set_caption("Adventure Game")
  black=(0,0,0)
  white=(255,255,255)
  img=pygame.image.load("/home/pi/Downloads/
sprite1.png")

  def sprite(x,y):
    gamewindow.blit(img, (x,y))

  x=(800*0.45)
  y=(600*0.8)

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

  for event in pygame.event.get():
    if event.type==pygame.QUIT:
      running=False
```



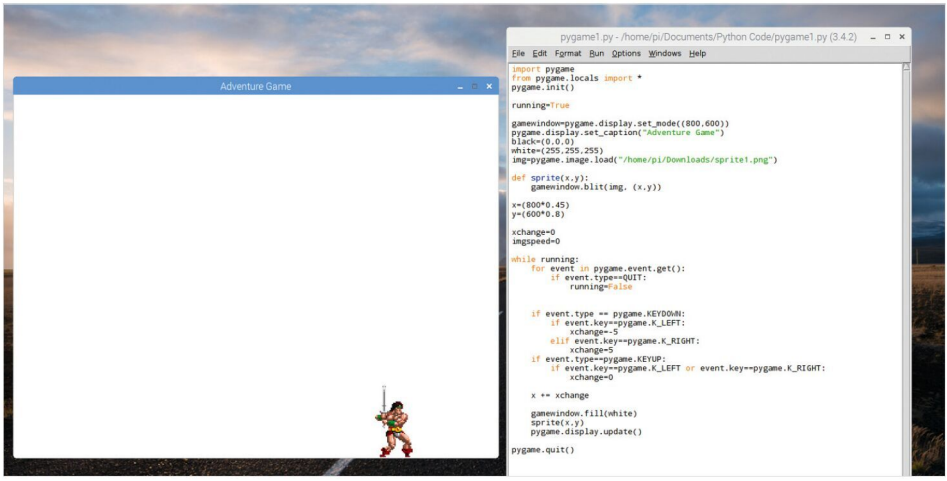
STEP 6

Let's quickly go through the code changes. We've defined two colours, black and white together with their respective RGB colour values. Next we've loaded the downloaded image called sprite1.png and allocated it to the variable img; and also defined a sprite function and the Blit function will allow us to eventually move the image.

```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))

    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()



    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

STEP 7 Now we can change the code around again, this time containing a movement option within the while loop, and adding the variables needed to move the sprite around the screen:

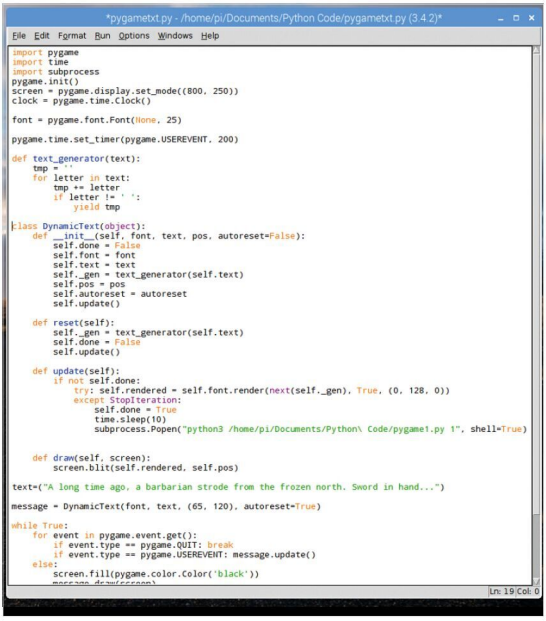
```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.
png")

def sprite(x,y):
  gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0

while running:
  for event in pygame.event.get():
    if event.type==QUIT:
      running=False

  if event.type == pygame.KEYDOWN:
    if event.key==pygame.K_LEFT:
      xchange=-5
    elif event.key==pygame.K_RIGHT:
      xchange=5
  if event.type==pygame.KEYUP:
    if event.key==pygame.K_LEFT or event
key==pygame.K_RIGHT:
      xchange=0

  x += xchange

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

pygame.quit()
```



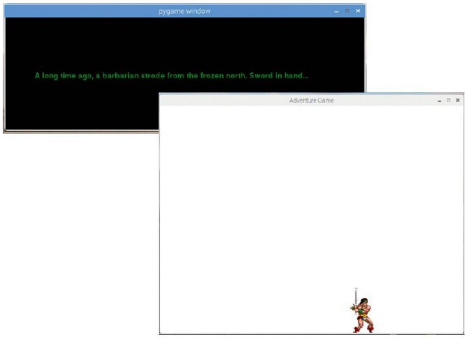
STEP 8 Copy the code down and using the left and right arrow keys on the keyboard you can move your sprite across the bottom of the screen. Now, it looks like you have the makings of a classic arcade 2D scroller in the works.

STEP 9 You can now implement a few additions and utilise some previous tutorial code. The new elements are the Subprocess module, of which one function allows us to launch a second Python script from within another; and we're going to create a New File called pygametxt.py:

```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
  tmp = ''
  for letter in text:
    tmp += letter
    if letter != ' ':
      yield tmp

class DynamicText(object):
  def __init__(self, font, text, pos,
autoreset=False):
    self.done = False
    self.font = font
    self.text = text
    self._gen = text_generator(self.text)
    self.pos = pos
    self.autoreset = autoreset
    self.update()

  def reset(self):
    self._gen = text_generator(self.text)
    self.done = False
    self.update()

  def update(self):
    if not self.done:
      try: self.rendered = self.font.
render(next(self._gen), True, (0, 128, 0))
      except StopIteration:
        self.done = True
        time.sleep(10)
        subprocess.Popen("python3 /home/pi/Documents/
Python\ Code/pygame1.py 1", shell=True)

  def draw(self, screen):
    screen.blit(self.rendered, self.pos)

text=("A long time ago, a barbarian strode from the
frozen north. Sword in hand...")

message = DynamicText(font, text, (65, 120),
autoreset=True)

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: break
    if event.type == pygame.USEREVENT: message.
update()
  else:
    screen.fill(pygame.color.Color('black'))
    message.draw(screen)
    pygame.display.flip()
    clock.tick(60)
    continue
  break

pygame.quit()
```



STEP 10 When you run this code it will display a long, narrow Pygame window with the intro text scrolling to the right. After a pause of ten seconds, it then launches the main game Python script where you can move the warrior sprite around. Overall the effect is quite good but there's always room for improvement.



Basic Animation

Python's modules make it relatively easy to create shapes, or display graphics and animate them accordingly. Animation though, can be a tricky element to get right in code. There are many different ways of achieving the same end result and we'll show you one such example here.

LIGHTS, CAMERA, ACTION

The Tkinter module is an ideal starting point for learning animation within Python. Naturally, there are better custom modules out there, but Tkinter does the job well enough to get a grasp on what's needed.

STEP 1

Let's make a bouncing ball animation. First, we will need to create a canvas (window) and the ball to animate:

```
from tkinter import *
import time

gui = Tk()
gui.geometry("800x600")
gui.title("Pi Animation")
canvas = Canvas(gui,
width=800,height=600,bg='white')
canvas.pack()

ball1 = canvas.create_oval(5,5,60,60, fill='red')

gui.mainloop()
```

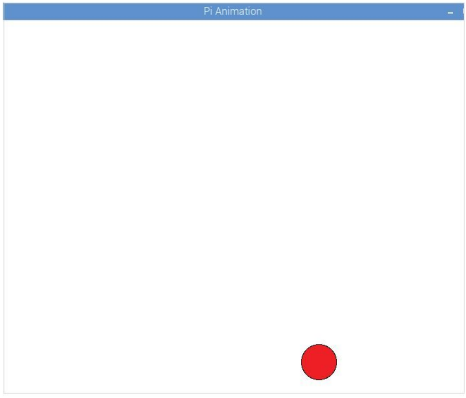
STEP 2

Save and Run the code. You will see a blank window appear, with a red ball sitting in the upper left corner of the window. While this is great, it's not very animated. Let's add the following code:

```
a = 5
b = 5

for x in range(0,100):
        canvas.move(ball1,a,b)
        gui.update()
        time.sleep(.01)
```

STEP 3

Insert the new code between the `ball1 = canvas.create_oval(5,5,60,60, fill='red')` line and the `gui.mainloop()` line. Save it and Run. You will now see the ball move from the top left corner of the animation window, down to the bottom right corner. You can alter the speed in which the ball traverses the window by altering the `time.sleep(.01)` line. Try (.05).



STEP 4

The `canvas.move(ball1,a,b)` line is the part that moves the ball from one corner to the other; obviously with both a and b equalling 5. We can change things around a bit already, such as the size and colour of the ball, with the line: `ball1 = canvas.create_oval(5,5,60,60, fill='red')` and we can change the values of a and b to something else.

```
ball1 = canvas.create_oval(7,7,60,60, fill='red')

a = 8
b = 3

for x in range(0,100):
    canvas.move(ball1,a,b)
    gui.update()
    time.sleep(.05)
```

STEP 5

Let's see if we can animate the ball so that it bounces around the window until you close the program.

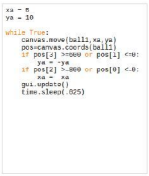
```
xa = 5
ya = 10

while True:
        canvas.move(ball1,xa,ya)
        pos=canvas.coords(ball1)
       if pos[3] >=600 or pos[1] <=0:
            ya = -ya
        if pos[2] >=800 or pos[0] <=0:
            xa = -xa
        gui.update()
       time.sleep(.025)
```

STEP 6

Remove the code you entered in Step 2 and insert the code from Step 5 in its place; again, between the `ball1 = canvas.create_oval(5,5,60,60, fill='red')` and the `gui.mainloop()` lines. Save the code and Run it as normal. If you've entered the code correctly, then you will see the red ball bounce off the edges of the window until you close the program.

STEP 7

The bouncing animation takes place within the `While True` loop. First, we have the values of xa and xy before the loop, both of 5 and 10. The `pos=canvas.coords(ball1)` line takes the value of the ball's location in the window. When it reaches the limits of the window, 800 or 600, it will make the values negative; moving the ball around the screen.



STEP 8

Pygame, however, is a much better module at producing higher-end animations. Begin by creating a New File and entering:

```
import pygame
from random import randrange

MAX_STARS  = 250
STAR_SPEED = 2

def init_stars(screen):
  """ Create the starfield """
  global stars
  stars = []
  for i in range(MAX_STARS):
    # A star is represented as a list with this
format:  [X,Y]
    star = [randrange(0,screen.get_width() - 1),
            randrange(0,screen.get_height() - 1)]
    stars.append(star)

def move_and_draw_stars(screen):
  """ Move and draw the stars """
  global stars
  for star in stars:
    star[1] += STAR_SPEED
    if star[1] >= screen.get_height():
      star[1] = 0
      star[0] = randrange(0,639)

    screen.set_at(star,(255,255,255))
```

STEP 9

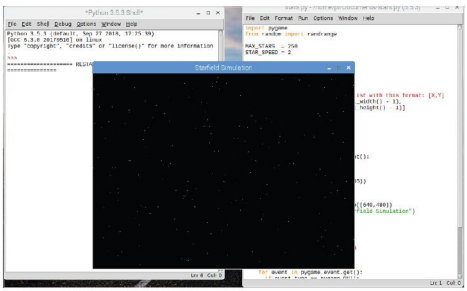
Now add the following:

```
def main():
  pygame.init()
  screen = pygame.display.set_mode((640,480))
  pygame.display.set_caption("Starfield
Simulation")
  clock = pygame.time.Clock()

  init_stars(screen)

  while True:
    # Lock the framerate at 50 FPS
    clock.tick(50)

    # Handle events
    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        return

    screen.fill((0,0,0))
    move_and_draw_stars(screen)
    pygame.display.flip()

if __name__ == "__main__":
  main()
```

```
def main():
  pygame.init()
  screen = pygame.display.set_mode((640,480))
  pygame.display.set_caption("Starfield Simulation")
  clock = pygame.time.Clock()

  init_stars(screen)

  while True:
    # Lock the framerate at 50 FPS
    clock.tick(50)

    # Handle events
    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        return

    screen.fill((0,0,0))
    move_and_draw_stars(screen)
    pygame.display.flip()

if __name__ == "__main__":
  main()
```

STEP 10

Save and Run the code. You will agree that the simulated starfield code looks quite impressive. Imagine this as the beginning of some game code, or even the start to a presentation? Using a combination of Pygame and Tkinter, your Python animations will look fantastic.



Create Your Own Modules

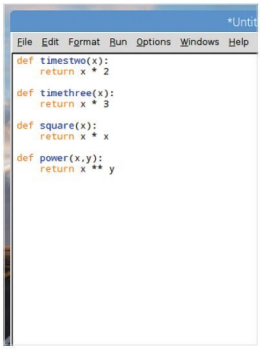
Large programs can be much easier to manage if you break them up into smaller parts and import the parts you need as modules. Learning to build your own modules also makes it easier to understand how they work.

BUILDING MODULES

Modules are Python files, containing code, that you save using a .py extension. These are then imported into Python using the now familiar import command.

STEP 1 Let's start by creating a set of basic mathematics functions. Multiply a number by two, three and square or raise a number to an exponent (power). Create a New File in the IDLE and enter:

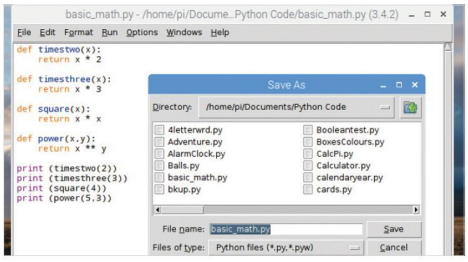
```
def timestwo(x):
        return x * 2

def timesthree(x):
        return x * 3

def square(x):
        return x * x

def power(x,y):
        return x ** y
```



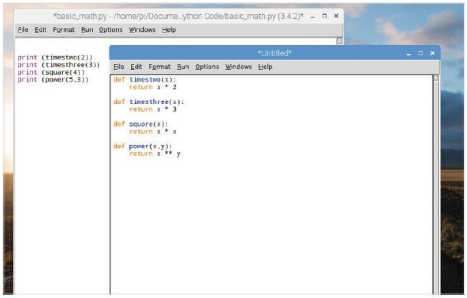
STEP 2 Under the above code, enter functions to call the code:

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

Save the program as basic_math.py and execute it to get the results.



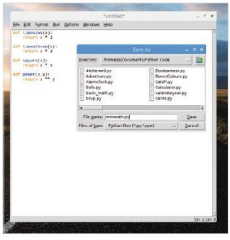
STEP 3 Now you're going to take the function definitions out of the program and into a separate file. Highlight the function definitions and choose Edit > Cut. Choose File > New File and use Edit > Paste in the new window. You now have two separate files, one with the function definitions, the other with the function calls.



STEP 4 If you now try and execute the basic_math.py code again, the error **'NameError: name 'timestwo' is not defined'** will be displayed. This is due to the code no longer having access to the function definitions.

```
Traceback (most recent call last):
  File "/home/pi/Documents/Python Code/basic_math.py", line 3, in <module>
    print (timestwo(2))
NameError: name 'timestwo' is not defined
>>>
```

STEP 5 Return to the newly created window containing the function definitions, and click File > Save As. Name this **minimath.py** and save it in the same location as the original **basic_math.py** program. Now close the minimath.py window, so the basic_math.py window is left open.

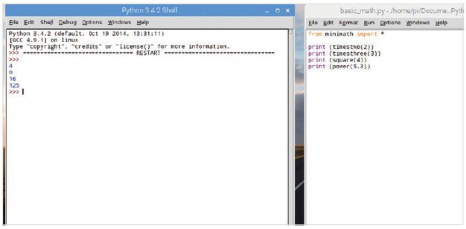


STEP 6

Back to the basic_math.py window: at the top of the code enter:

```
from minimath import *
```

This will import the function definitions as a module. Press F5 to save and execute the program to see it in action.



STEP 7

You can now use the code further to make the program a little more advanced, utilising the newly created module to its full. Include some user interaction. Start by creating a basic menu the user can choose from:

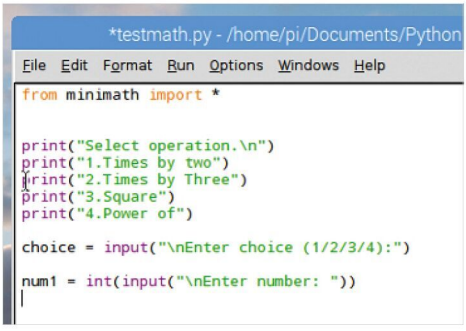
```
print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")
```

STEP 8

Now we can add the user input to get the number the code will work on:

```
num1 = int(input("\nEnter number: "))
```

This will save the user-entered number as the variable num1.



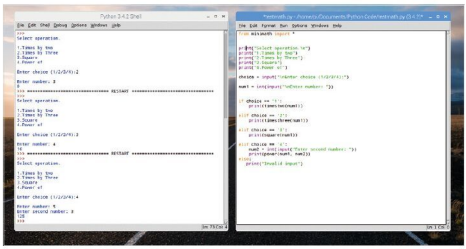
STEP 9

Finally, you can now create a range of if statements to determine what to do with the number and utilise the newly created function definitions:

```
if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

STEP 10

Note that for the last available options, the Power of choice, we've added a second variable, num2. This passes a second number through the function definition called power. Save and execute the program to see it in action.



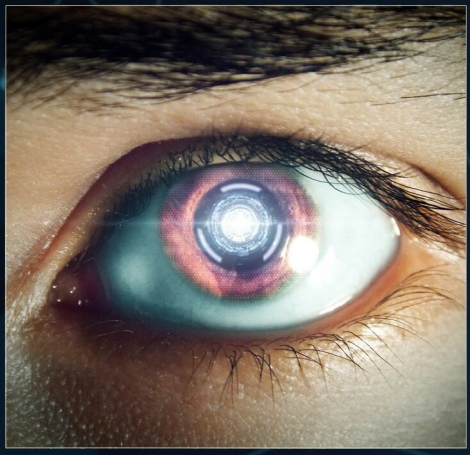
Python in Focus: Artificial Intelligence

Artificial Intelligence (AI) and Machine Learning (ML) are the new hot topics of the IT industry. AI is fast becoming the working science fiction that it has been portrayed as in the past, and behind it is Python.

Despite how close AI and ML are, there are distinct differences between the two technologies. AI refers to the study of how to train a computer to accomplice the things that humans can do significantly better and faster. Whereas, ML is the ability for a computer to learn from its experiences, so that the outcome and performance will eventually become more accurate and accomplished.

While different, they are both essentially discussing the same element: training a system to learn and do things independently. Where AI is said to lead to wisdom, ML reportedly leads to knowledge and, thanks to Python, that gap is getting closer every day.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```



APPLICATIONS

Both AI and ML are hugely present in today's technology. Where, just a few years ago, most of us associated AI with the rise of a super-intelligent legion of killer robots, nowadays you'd be amazed at the numerous examples of AI in your house, and even being carried around with you.

Let's begin with the obvious use of AI and ML, the smartphone. These devices have infiltrated most of our modern world, with global coverage reaching 5.5 billion for 2019 and set to rise to over 6 billion by the end of 2020, it's little surprise to discover that AI and ML are advancing in leaps and bounds.

With nearly all of the population of humanity within reach of a smartphone, the coding behind these devices has been developed to take individuals into account. These devices are designed to learn what the user requires, or uses, the device for. Common numbers called are pushed to the top of the list, in-app and in-game advertising is moulded around our browser and search preferences, as well as other apps we've installed in the past. And even our voices, fingerprints and faces are stored and analysed by AI and ML in order to recognise who we are.

DIGITAL ASSISTANTS

The rise of digital assistants has been one of the kick-starters of AI and ML programming. Siri, Cortana, Alexa and Google Assistant are all coded using Python, and are designed to listen, learn and respond to what we ask of them. With Python, this level of AI is surprisingly simple, thanks to the many libraries and customisation of the language. These frameworks make creating AI and ML easy for intelligent coders, cutting down on the development time in other languages and, thanks to Python's easy to read code and complex algorithms, these developers can devote significant time to improving the performance and accuracy of AI.

Every time we ask one of these digital assistants for something, the Python-driven AI code is reading our voice, determining what it is we're asking by plucking out key words and acting on them. If we ask for a thirty second countdown, it'll start the device's stopwatch function; if we ask for dinner suggestions, it'll open a specific set of web pages, and if we ask it to play some music, it'll interrogate the available music apps to select what it is we wanted. All the time, the AI code is being trained to listen more intently, while the ML is learning from the AI results so that its accuracy is improved for future questions and requests.

BEYOND THE SMARTPHONE

Consider Google, social media and the content you look up. How many times have you entered a search string into Google, such as car parts for a Mk1 Ford Escort and, when you've opened Facebook, you suddenly find a group suggestion of Ford Escort owners? That's AI and ML injecting themselves into your everyday computing tasks.

Another example of AI and ML working together is Gmail's recent addition of suggested completions for sentences you are typing. If you frequently sign off with 'See you soon', or 'All the best', then typing 'See' or 'All' will prompt the ML side of the equation to autofill the remainder of the words for you. All the time, the ML is learning while the AI is telling it what to improve on.

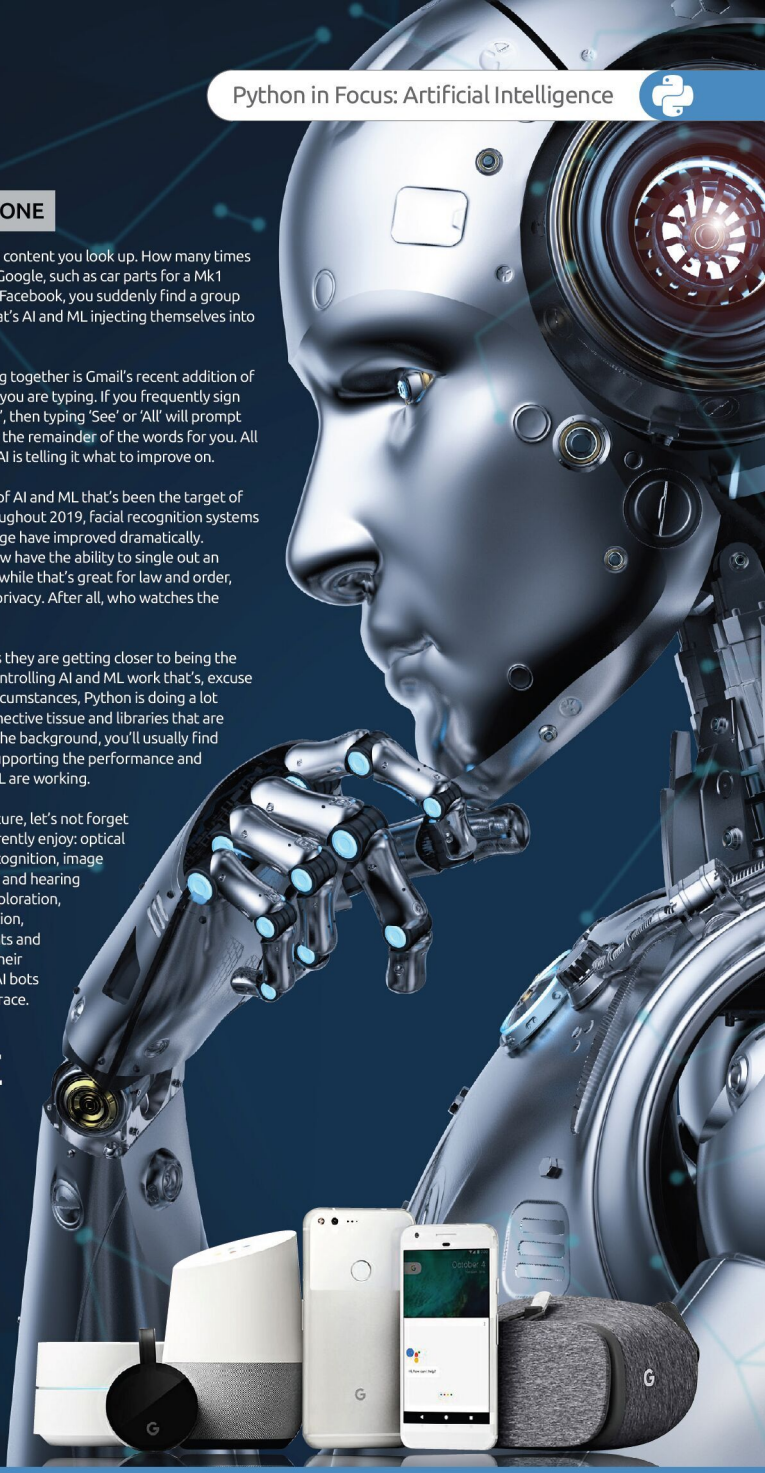
Facial recognition is another element of AI and ML that's been the target of the popular press for some time. Throughout 2019, facial recognition systems on both smartphones and CCTV footage have improved dramatically. Agencies controlling this level of AI now have the ability to single out an individual from a crowded street and, while that's great for law and order, it does pose a potential threat to our privacy. After all, who watches the watcher?

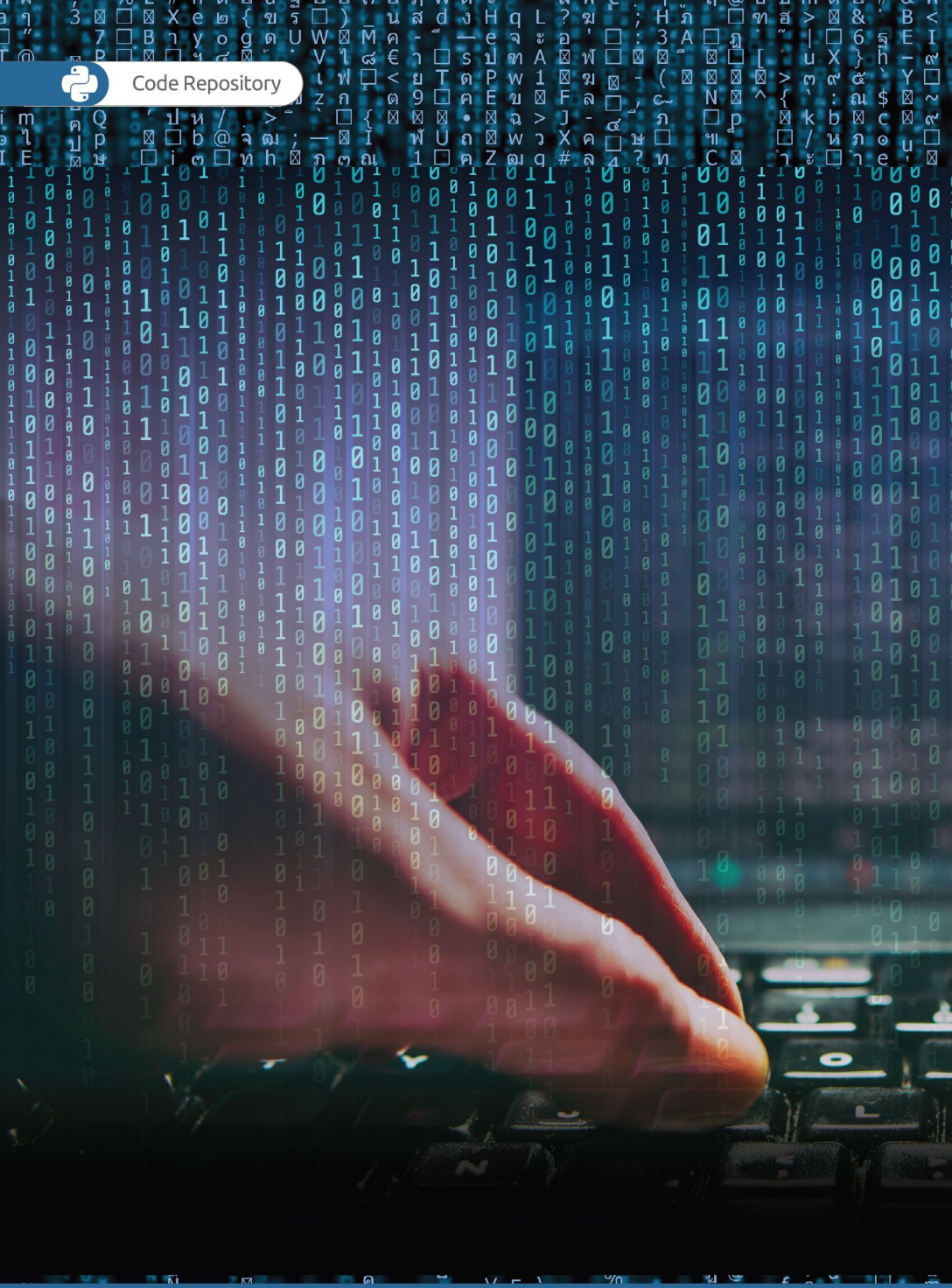
Tesla's work on self-driving cars means they are getting closer to being the norm, and it's Python along with its controlling AI and ML work that's, excuse the pun, driving it forward. In these circumstances, Python is doing a lot of the heavy lifting, providing the connective tissue and libraries that are designed to implement AI and ML. In the background, you'll usually find C++, or some other language, that's supporting the performance and overall program in which the AI and ML are working.

While it's easy to portray a bleak AI future, let's not forget the many great instances of AI we currently enjoy: optical character recognition, handwriting recognition, image processing, helping people with visual and hearing disabilities, advancements in space exploration, engineering improvements, conservation, pharmaceutical and drug improvements and greater freedom for those limited in their ability to travel. It's not all about two AI bots arguing about eliminating the human race.

THE FUTURE OF AI

Whether we'll end up creating true AI, killer robots and self-aware androids is up for debate. There are plenty of arguments for and against the evolution of AI, with many believing that AI will be the worst possible future humans can create – worse even than nuclear war. For the moment, however, we're at the early stages of AI development, but with Python's continual advancements and improved libraries, it may not be too long before we've got an AI system that's getting better by the hour.




Code Repository

Code Repository

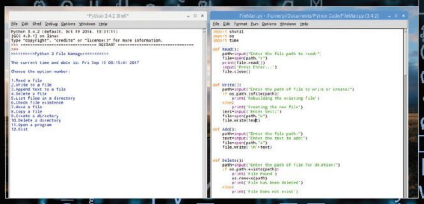
We've included a vast Python code repository for you to freely use in your own programs. There's plenty in here to help you create a superb piece of programming, or extend your project ideas.

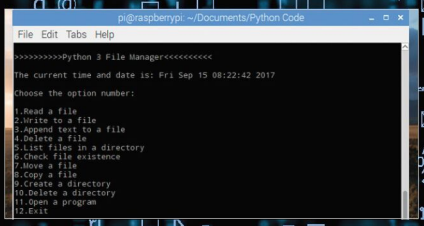
We've got code for making backups of your files and folders, number guessing games, random number generators, Google search code, game code, animation code, graphics code, text adventure code and even code that plays music stored on your computer. We've broken down some of the newer, and extended, concepts of the code to help you better understand what's going on. This way you can easily adapt it to your own uses.

This is an excellent resource that you won't find in any other Python book. So use it, take it apart, adapt it to your own programs and see what you can create.

Python File Manager

This file manager program displays a list of options that allow you to read a file, write to a file, append to a file, delete a file, list the contents of a directory and much more. It's remarkably easy to edit and insert into your own code, or add to.





1 This part of the code imports the necessary modules. The OS and Subprocess modules deal with the operating system elements of the program.

2 Each def XXX() functions store the code for each of the menu's options. Once the code within the function is complete, the code returns to the main menu for another option.

3 This is part of the code that checks to see what OS the user is running. In Windows the CLS command clears the screen, whereas in Linux and macOS, the Clear command wipes the screen. If the code tries to run CLS when being used in Linux or macOS, an error occurs, which then prompts it to run the Clear command instead.

4 These are the options, from 1 to 12. Each executes the appropriate function when the relevant number is entered.

FILEMAN.PY

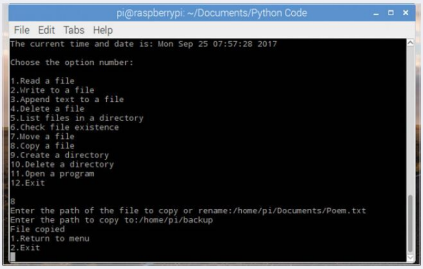
Copy the code below into a New > File and save it as FileMan.py. Once executed it will display the program title, along with the current time and date and the available options.

```
import shutil
import os
import time
import subprocess

def Read():
   path=input("Enter the file path to read:")
   file=open(path,"r")
   print(file.read())
   input('Press Enter...')
   file.close()

def Write():
   path=input("Enter the path of file to write or create:")
   if os.path.isfile(path):
      print('Rebuilding the existing file')
   else:
      print('Creating the new file')
   text=input("Enter text:")
   file=open(path,"w")
   file.write(text)

def Add():
   path=input("Enter the file path:")
   text=input("Enter the text to add:")
   file=open(path,"a")
   file.write('\n'+text)

def Delete():
   path=input("Enter the path of file for deletion:")
   if os.path.exists(path):
      print('File Found')
      os.remove(path)
      print('File has been deleted')
   else:
      print('File Does not exist')

def Dirlist():
   path=input("Enter the Directory path to display:")
   sortlist=sorted(os.listdir(path))
   i=0
   while(i<len(sortlist)):
      print(sortlist[i]+'\n')
      i+=1

def Check():
   fp=int(input('Check existence of \n1.File \n2.
   Directory\n'))
      if fp==1:
         path=input("Enter the file path:")
         os.path.isfile(path)
```

```
        if os.path.isfile(path)==True:
            print('File Found')
        else:
            print('File not found')
    if fp==2:
        path=input("Enter the directory path:")
        os.path.isdir(path)
        if os.path.isdir(path)==False:
            print('Directory Found')
        else:
            print('Directory Not Found')

def Move():
    path1=input('Enter the source path of file to move:')
    mr=int(input('1.Rename \n2.Move \n'))
    if mr==1:
        path2=input('Enter the destination path and file name:')
        shutil.move(path1,path2)
        print('File renamed')
    if mr==2:
        path2=input('Enter the path to move:')
        shutil.move(path1,path2)
        print('File moved')

def Copy():
    path1=input('Enter the path of the file to copy or rename:')
    path2=input('Enter the path to copy to:')
    shutil.copy(path1,path2)
    print('File copied')

def Makedir():
    path=input("Enter the directory name with path to make
    \neg. C:\\Hello\\Newdir \nWhere Newdir is new
    directory:")
    os.makedirs(path)
    print('Directory Created')

def Removedir():
    path=input('Enter the path of Directory:')
    treedir=int(input('1.Deleted Directory \n2.Delete
    Directory Tree \n3.Exit \n'))
    if treedir==1:
        os.rmdir(path)
    if treedir==2:
        shutil.rmtree(path)
        print('Directory Deleted')
    if treedir==3:
        exit()

def Openfile():
    path=input('Enter the path of program:')
    try:
        os.startfile(path)
    except:
        print('File not found')

run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
    print('The current time and date is:',time.asctime())
    print('\nChoose the option number: \n')
    dec=int(input('''1.Read a file
2.Write to a file
3.Append text to a file
4.Delete a file
5.List files in a directory
6.Check file existence
7.Move a file
8.Copy a file
9.Create a directory
10.Delete a directory
11.Open a program
12.Exit

'''))
    if dec==1:
        Read()
    if dec==2:
        Write()
    if dec==3:
        Add()
    if dec==4:
        Delete()
    if dec==5:
        Dirlist()
    if dec==6:
        Check()
    if dec==7:
        Move()
    if dec==8:
        Copy()
    if dec==9:
        Makedir()
    if dec==10:
        Removedir()
    if dec==11:
        Openfile()
    if dec==12:
        exit()
    run=int(input("1.Return to menu\n2.Exit \n"))
    if run==2:
        exit()
```

3

4



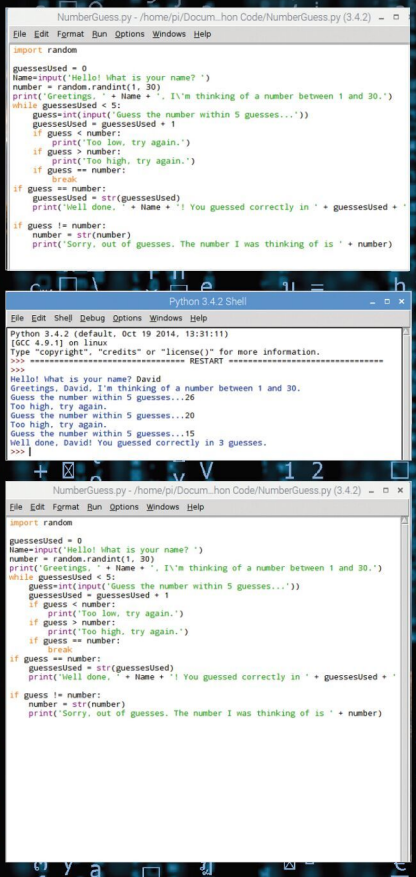
Imports

There are three modules to import here: Shutil, OS and Time. The first two deal with the operating system and file management and manipulation; and the Time module simply displays the current time and date.

Note how we've included a try and except block to check if the user is running the code on a Linux system or Windows. Windows uses CLS to clear the screen, while Linux uses clear. The try block should work well enough but it's a point of possible improvement depending on your own system.

Number Guessing Game

This is a simple little piece of code but it makes good use of the Random module, print and input, and a while loop. The number of guesses can be increased from 5 and the random number range can easily be altered too.



NUMBERGUESS.PY

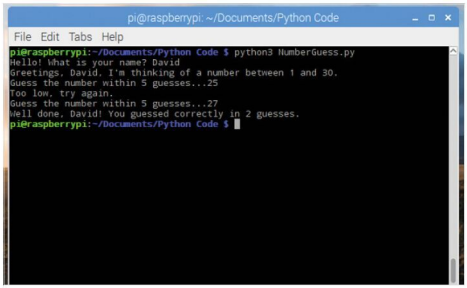
Copy the code and see if you can beat the computer within five guesses. It's an interesting bit of code that can be quite handy when your implementing a combination of the Random module alongside a while loop.

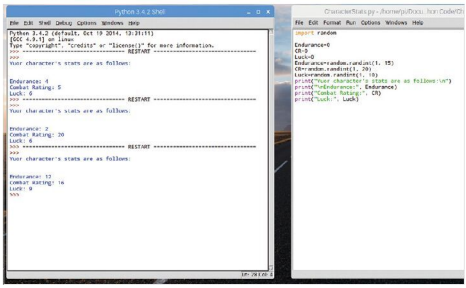
```
import random

guessesUsed = 0
Name=input('Hello! What is your name? ')
number = random.randint(1, 30)
print('Greetings, ' + Name + ', I\'m thinking of a
number between 1 and 30.')
while guessesUsed < 5:
   guess=int(input('Guess the number within 5 guesses...'))
   guessesUsed = guessesUsed + 1
   if guess < number:
      print('Too low, try again.')
   if guess > number:
      print('Too high, try again.')
   if guess == number:
      break
if guess == number:
   guessesUsed = str(guessesUsed)
   print('Well done, ' + Name + '! You guessed
   correctly in ' + guessesUsed + ' guesses.')

if guess != number:
   number = str(number)
   print('Sorry, out of guesses. The number I was
   thinking of is ' + number)
```

1 Although this is a reasonably easy to follow program, there are some elements to the code that are worth pointing out. To begin with, you need to import the Random module, as you're using random numbers within the code.

2 This section of the code creates the variables for the number of guesses used, along with the name of the player, and also sets up the random number between 1 and 30. If you want a wider range of random number selection, then increase the **number=random.randint(1, 30)** end value of 30; don't make it too high though or the player will never be able to guess it. If the player guesses too low or too high, they are given the appropriate output and asked to try again, while the number of guesses is less than five. You can also increase the number of guesses from 5 by altering the **while guessesUsed < 5:** value.

3 If the player guessed the correct number then they are given a 'well done' output, along with how many guesses they used up. If the player runs out of guesses, then the game over output is displayed instead, along with revealing the number the computer was thinking of. Remember, if you do alter the values of the random number chosen by the computer, or the number of guesses the player can take, then along with the variable values, you also need to amend the instructions given in the print statements at the start of the code.





Code Improvements

Since this is such as simple script to apply to a situation, there's plenty of room to mess around with it and make it more interesting. Perhaps you can include an option to take score, the best out of three rounds. Maybe an elaborate way to congratulate the player for getting a 'hole in one' correct guess on their first try.

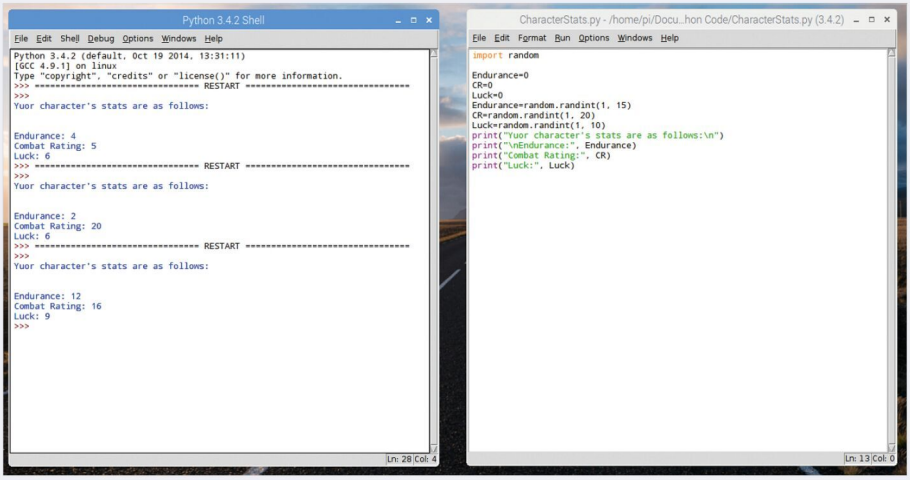
Moreover, the number guessing game code does offer some room for implementing into your code in a different manner. What we mean by this is, the code can be used to retrieve a random number between a range, which in turn can give you the start of a character creation defined function within an adventure game.

Imagine the start of a text adventure written in Python, where the player names their character. The next step is to roll the virtual random dice to decide what that character's combat rating, strength, endurance and luck values are. These can then be carried forward into the game under a set of variables that can be reduced or increased depending on the circumstances the player's character ends up in.

For example, as per the screenshot provided, you could use something along the lines of:

```
Endurance=0
CR=0
Luck=0
Endurance = random.randint(1, 15)
CR = random.randint(1, 20)
Luck = random.randint(1, 10)
Print("Your character's stats are as follows:\n")
Print("Endurance:", Endurance)
Print("Combat Rating:", CR)
Print("Luck:", Luck)
```

The player can then decide to either stick with their roll or try again for the hope of better values being picked. There's ample ways in which to implement this code into a basic adventure game.



Random Number Generator

User input and the ability to manipulate that input are important elements with any programming language. It's what separates a good program from a great program, one that allows the user to interact and see the results of that interaction.

RNDNUMGEN.PY

It might be simple but this little piece of code will ask the user for two sets of numbers, a start and a finish. The code will then pluck out a random number between the two sets and display it.

```
from random import *

print("\n>>>>>>>>>>Random Number Generator<<<<<<<<<<\n")
nmb1=int(input("Enter the start number: "))
nmb2=int(input("Enter the last number: "))

x = randint(nmb1, nmb2)
print("\nThe random number between",nmb1,"and",nmb2,"is:\n")
print(x)
```

More Input

While an easy code to follow, it could be more interesting if you prompt the user for more input. Perhaps you can provide them with addition, subtraction, multiplication elements with their numbers. If you're feeling clever, see if you can pass the code through a Tkinter window or even the Ticker window that's available on Page 16.

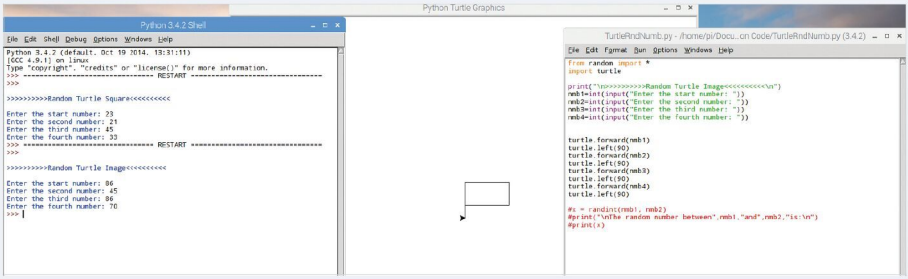
Furthermore, the core of the code can be used in a text adventure game, where the character fights something and their health, along with the enemy's, is reduced by a random number. This can be mixed with the previous code from Page 32's Number Guessing Game, where we defined the stats for the adventure game's character.

You can also introduce the Turtle module into the code and perhaps set some defined rules for drawing a shape, object or something based on a user inputted random value from a range of numbers. It takes a little working out but the effect is certainly really interesting.

For example, the code could be edited to this:

```
from random import *
import turtle

print("\n>>>>>>>>>>Random Turtle Image<<<<<<<<<<\n")
nmb1=int(input("Enter the start number: "))
nmb2=int(input("Enter the second number: "))
nmb3=int(input("Enter the third number: "))
nmb4=int(input("Enter the fourth number: "))



turtle.forward(nmb1)
turtle.left(90)
turtle.forward(nmb2)
turtle.left(90)
turtle.forward(nmb3)
turtle.left(90)
turtle.forward(nmb4)
turtle.left(90)
```

Whilst it's a little rough around the edges, you can easily make it more suitable.



Random Password Generator

We're always being told that our passwords aren't secure enough; well here's a solution for you to implement into your own future programs. The random password generator code below will create a 12-letter string of words (both cases) and numbers each time it's executed.

RNDPASSWORD.PY

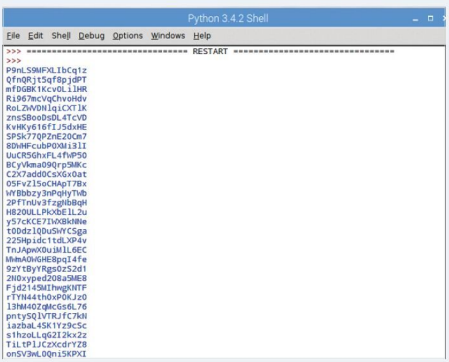
Copy the code and run it; each time you'll get a random string of characters that can easily be used as a secure password which will be very difficult for a password cracker to hack.

```
import string
import random

def randompassword():
   chars=string.ascii_uppercase + string.ascii_
   lowercase + string.digits
   size= 8
   return ''.join(random.choice(chars) for x in
   range(size,20))

print(randompassword())
```

Secure Passwords

There's plenty you can do to modify this code and improve it further. For one, you can increase the number of characters the generated password displays and perhaps you can include special characters too, such as signs and symbols. Then, you can output the chosen password to a file, then securely compress it using the previous random number generator as a file password and send it to a user for their new password.

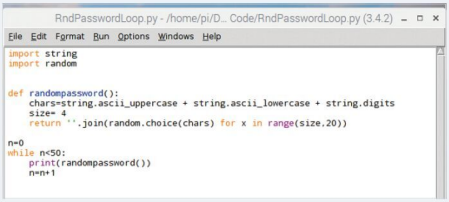
An interesting aspect to this code is the ability to introduce a loop and print any number of random passwords. Let's assume you have a list of 50 users for a company and you're in charge of generating a random password for them each month.



Adding a loop to print a password fifty times is extremely easy, for example:

```
import string
import random


def randompassword():
   chars=string.ascii_uppercase + string.ascii_
   lowercase + string.digits
   size= 4
   return ''.join(random.choice(chars) for x in
   range(size,20))

n=0
while n<50:
   print(randompassword())
   n=n+1
```

This will output fifty random passwords based on the previous random selection of characters.



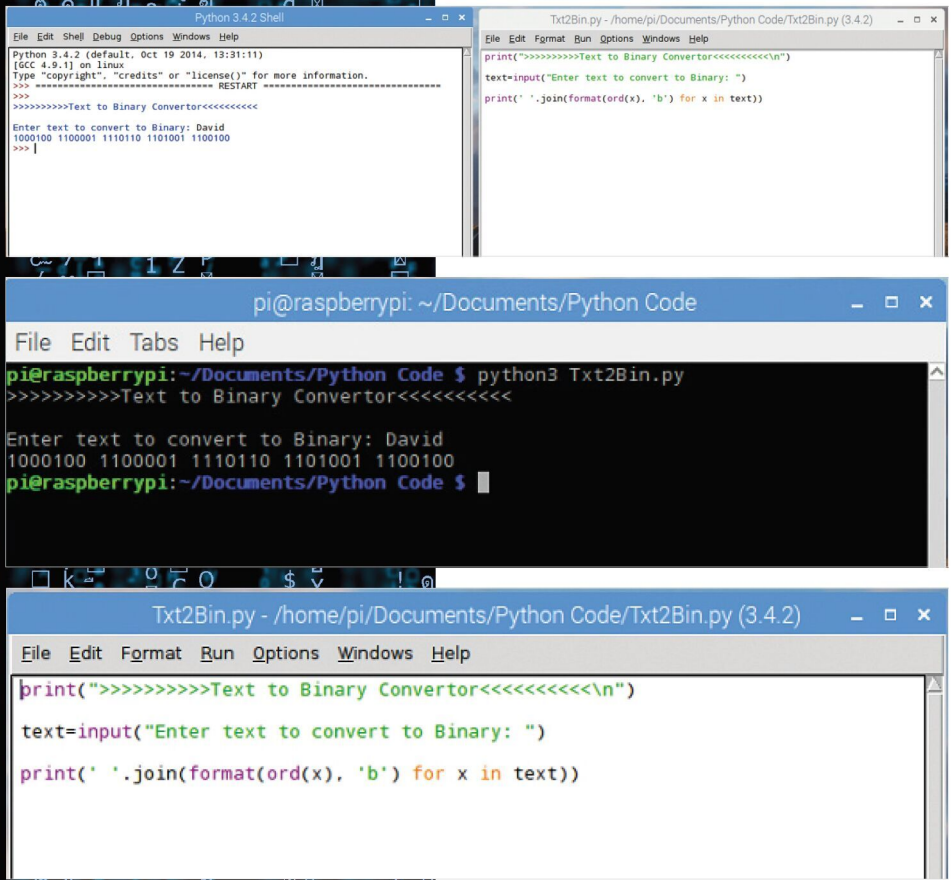
Text to Binary Convertor

While it may not seem too exciting, this text to binary convertor is actually quite good fun. It also only uses two lines of code, so it's extremely easy to insert into your own script.

TXT2BIN.PY

Naturally we're using the format function to convert the user's entered text string into its binary equivalent. If you want to check its accuracy, you can plug the binary into an online convertor.

```
text=input("Enter text to convert to Binary: ")

print(' '.join(format(ord(x), 'b') for x in text))
```



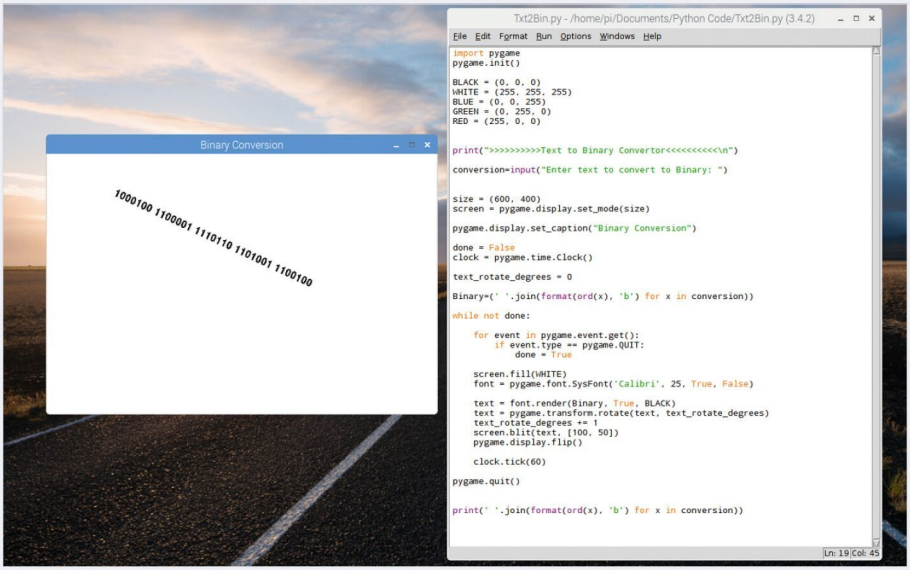
1000010 1101001 1101110 1100001 1110010 1111001

The text to binary convertor does offer some room for improvement and enhancement. There are many uses: it could be utilised in a password or secret word script, as part of an adventure game or just a novel way to display someone's name.

With regards to improvements, you could display the binary conversion in a Pygame window, using the animated text options from page 18. You could also ask the user if they wanted to have another go, or even ask if they wanted the binary output to be saved to a file.

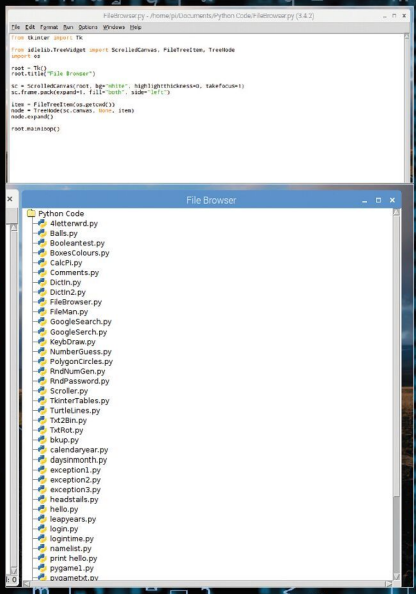
With regards to rendering the outputted binary conversion to a Pygame window, complete with rotating text, you can use:

```
import pygame
pygame.init()

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)

print(">>>>>>>>>>Text to Binary Convertor<<<<<<<<<<\n")

conversion=input("Enter text to convert to Binary: ")

size = (600, 400)
screen = pygame.display.set_mode(size)

pygame.display.set_caption("Binary Conversion")

done = False
clock = pygame.time.Clock()

text_rotate_degrees = 0

Binary=(' '.join(format(ord(x), 'b') for x
   in conversion))

while not done:

   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         done = True

   screen.fill(WHITE)
   font = pygame.font.SysFont('Calibri', 25, True, False)

   text = font.render(Binary, True, BLACK)
   text = pygame.transform.rotate(text, text_
   rotate_degrees)
   text_rotate_degrees += 1
   screen.blit(text, [100, 50])
   pygame.display.flip()

   clock.tick(60)

pygame.quit()

print(' '.join(format(ord(x), 'b') for x in conversion))
```



Basic GUI File Browser

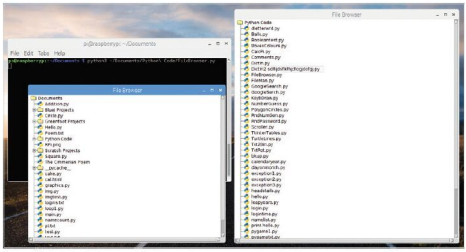
Here's a helpful and interesting piece of code. It's an extremely basic file browser that's presented in a graphical user interface using the Tkinter module. There's a lot you can learn from this code and implement into your own programs.

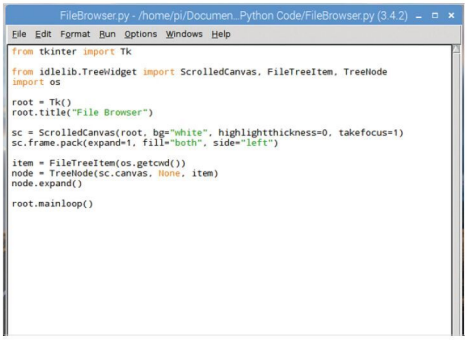


FILEBROWSER.PY

Tkinter is the main module in use here but we're also using idlelib, so you may need to pip install any extras if the dependencies fail when you execute the code.

```
from tkinter import Tk

from idlelib.TreeWidget import ScrolledCanvas,
FileTreeItem, TreeNode
import os

root = Tk()
root.title("File Browser")

sc = ScrolledCanvas(root, bg="white",
highlightthickness=0, takefocus=1)
sc.frame.pack(expand=1, fill="both", side="left")

item = FileTreeItem(os.getcwd())
node = TreeNode(sc.canvas, None, item)
node.expand()

root.mainloop()
```





Advanced Filing

When executed, the code will display the current directory's contents. If you want to see the contents of another directory, you can run the code from a command line within the chosen directory; just remember to call the code from where it's located on your system, as per the second screenshot. You can also double-click any of the file names shown in the directory tree and rename them.

This is an interesting piece of code and one that you can insert into your own programs. You can extend the code to include a user specified directory to browse, perhaps your own unique file icons too. If you're using Linux, create an alias to execute the code and then you can run it from wherever you are in the system.

Windows users may have some trouble with the above code, an alternative can be achieved by using the following:

```
from tkinter import *
from tkinter import ttk
from tkinter.filedialog import askopenfilename

root = Tk(  )

def OpenFile():
   name = askopenfilename(initialdir="C:/",
                  filetypes =(("Text File", "*.txt"),("All
                  Files","*.*")),
                  title = "Choose a file."
                              )
   print (name)
   try:
      with open(name,'r') as UseFile:
         print(UseFile.read())
   except:
      print("No files opened")


Title = root.title( "File Opener")
label = ttk.Label(root, text ="File
Open",foreground="red",font=("Helvetica", 16))
label.pack()


menu = Menu(root)
root.config(menu=menu)

file = Menu(menu)

file.add _ command(label = 'Open', command = OpenFile)
file.add _ command(label = 'Exit', command =
lambda:exit())

menu.add _ cascade(label = 'File', menu = file)

root.mainloop()
```

It's not quite the same but this code allows you to open files in your system via the familiar Windows Explorer. It's worth experimenting with to see what you can do with it.

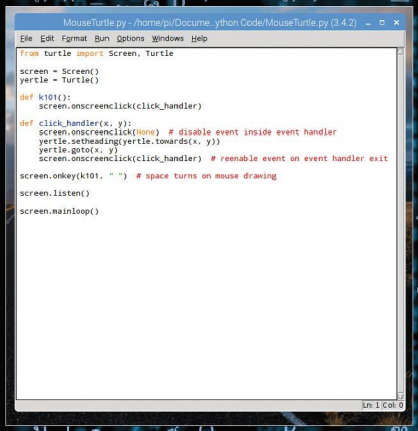
filetest1.py - C:\Users\david\Documents\Python\filetest1.py (3.6.2)

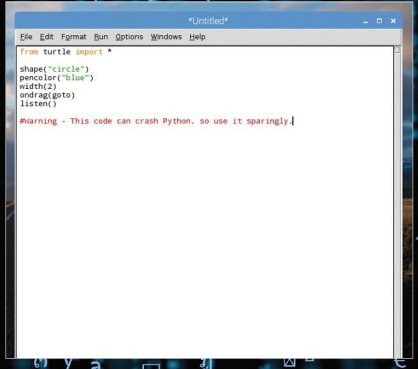
File Edit Format Run Options Window Help

```
from tkinter import *
from tkinter import ttk
from tkinter.filedialog import askopenfilename

root = Tk(  )

def OpenFile():
    name = askopenfilename(initialdir="C:/",
                           filetypes =(("Text File", "*.txt"),("All Files","*.*")),
                           title = "Choose a file."
                           )
    print (name)
    try:
        with open(name,'r') as UseFile:
            print(UseFile.read())
    except:
        print("No files opened")


Title = root.title( "File Opener")
label = ttk.Label(root, text ="File Open",foreground="red",font=("Helvetica", 16))
label.pack()


menu = Menu(root)
root.config(menu=menu)

file = Menu(menu)

file.add_command(label = 'Open', command = OpenFile)
file.add_command(label = 'Exit', command = lambda:exit())

menu.add_cascade(label = 'File', menu = file)


root.mainloop()
```

Mouse Controlled Turtle

We've already seen the Turtle module being controlled by the user via the keyboard but now we thought we'd see how the user can use their mouse as a drawing tool within Python. We have two possible code examples here, pick which works best for you.





MOUSETURTLE.PY

The first piece of code presents the standard Turtle window. Press Space and then click anywhere on the screen for the Turtle to draw to the mouse pointer. The second allows you to click the Turtle and drag it around the screen; but be warned, it can crash Python.

1st Code Example:

```
from turtle import Screen, Turtle

screen = Screen()
yertle = Turtle()

def k101():
   screen.onscreenclick(click_handler)

def click_handler(x, y):
   screen.onscreenclick(None)  # disable event inside
   event handler
   yertle.setheading(yertle.towards(x, y))
   yertle.goto(x, y)
   screen.onscreenclick(click_handler)  # reenable
   event on event handler exit

screen.onkey(k101, " ")  # space turns on mouse drawing

screen.listen()

screen.mainloop()
```

2nd Code Example:

```
from turtle import *
shape("circle")
pencolor("blue")
width(2)
ondrag(goto)
listen()
```

Ninja TurtleMouse

This code utilises some interesting skills. Obviously it will stretch your Python Turtle skills to come up with any improvements, which is great, but it could make for a nice piece of code to insert into something a young child will use. Therefore it can be a fantastic project for a younger person to get their teeth into; or perhaps even as part of a game where the main character is tasked to draw a skull and crossbones or something similar.

Python Alarm Clock

Ever taken a quick break from working at the computer, then suddenly realised many minutes later that you've spent all that time on Facebook? Introducing the Python alarm clock code, where you can drop into the command prompt and tell the code how many minutes until the alarm goes off.

ALARMCLOCK.PY

This code is designed for use in the command prompt, be that Windows, Linux or macOS. There are some instructions on how to use it in the main print section but essentially it's: python3 AlarmClock.py 10 (to go off in ten minutes).

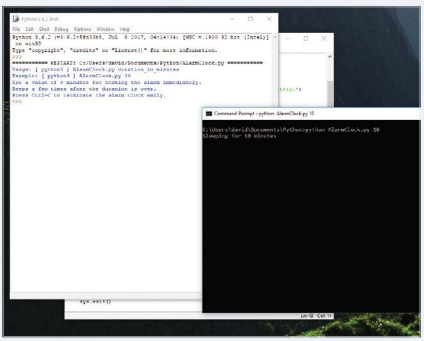
```
import sys
import string
from time import sleep

sa = sys.argv
lsa = len(sys.argv)
if lsa != 2:
   print ("Usage: [ python3 ] AlarmClock.py duration _
   in _ minutes")
   print ("Example: [ python3 ] AlarmClock.py 10")
   print ("Use a value of 0 minutes for testing the
   alarm immediately.")
   print ("Beeps a few times after the duration is over.")
   print ("Press Ctrl-C to terminate the alarm
   clock early.")
   sys.exit(1)

try:
   minutes = int(sa[1])
except ValueError:
   print ("Invalid numeric value (%s) for minutes" % sa[1])
   print ("Should be an integer >= 0")
   sys.exit(1)

if minutes < 0:
   print ("Invalid value for minutes, should be >= 0")
   sys.exit(1)

seconds = minutes * 60

if minutes == 1:
   unit _ word = " minute"
else:
   unit _ word = " minutes"
```

```
try:
   if minutes > 0:
      print ("Sleeping for " + str(minutes) + unit _ word)
      sleep(seconds)
   print ("Wake up")
   for i in range(5):
      print (chr(7)),
      sleep(1)
except KeyboardInterrupt:
   print ("Interrupted by user")
   sys.exit(1)
```

Wakey Wakey

There's some good use of try and except blocks here, alongside some other useful loops that can help you get a firmer understanding of how they work in Python. The code itself can be used in a variety of ways: in a game where something happens after a set amount of time or simply as a handy desktop alarm clock for your tea break.

Linux users, try making the alarm clock code into an alias, so you can run a simple command to execute it. Then, why not integrate a user input at the beginning to ask the user for the length of time they want until the alarm goes off, rather than having to include it in the command line.



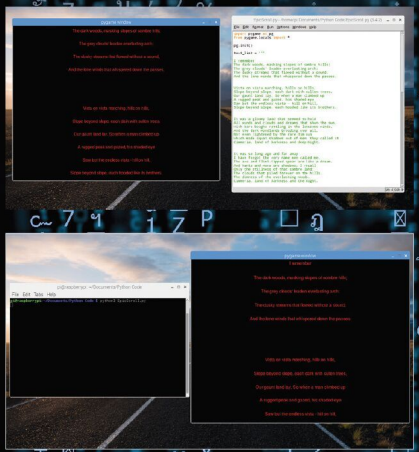
Windows users, if Python 3 is the only version installed on your system then you will need to execute the code without adding the 3 to the end of the Python command. For example:

```
python AlarmClock.py 10
```

Again, you could easily incorporate this into a Windows batch file and even set a schedule to activate the alarm at certain times of the day.

Vertically Scrolling Text

What's not to like about vertically scrolling text? Its uses are many: the beginning of a game or introduction to something epic, like the beginning of every Star Wars movie; a list of credits at the end of something, such as a Python presentation. The list goes on.



EPICSCROLL.PY

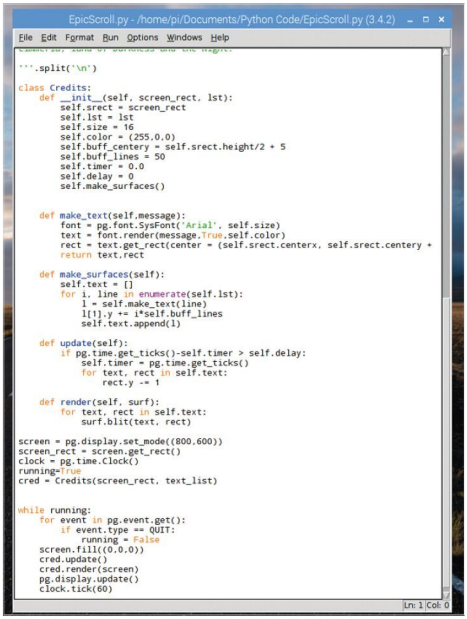
We've used the poem Cimmeria by Robert E. Howard for the code's scrolling text, along with a dramatic black background and red text. We think you'll agree, it's quite epic.

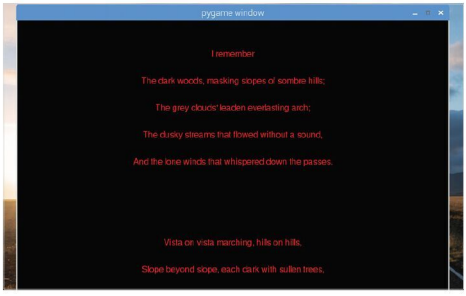
```
import pygame as pg
from pygame.locals import *

pg.init()

text_list = '''

I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


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


It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.


It was so long ago and far away
I have forgot the very name men called me.
The axe and flint-tipped spear are like a dream,
And hunts and wars are shadows. I recall
Only the stillness of that sombre land;
The clouds that piled forever on the hills,
The dimness of the everlasting woods.
Cimmeria, land of Darkness and the Night.


Oh, soul of mine, born out of shadowed hills,
To clouds and winds and ghosts that shun the sun,
How many deaths shall serve to break at last
This heritage which wraps me in the grey
Apparel of ghosts? I search my heart and find
Cimmeria, land of Darkness and the Night!

'''.split('\n')
```

```
class Credits:
   def __init__(self, screen_rect, lst):
      self.srect = screen_rect
      self.lst = lst
      self.size = 16
      self.color = (255,0,0)
      self.buff_centery = self.srect.height/2 + 5
      self.buff_lines = 50
      self.timer = 0.0
      self.delay = 0
      self.make_surfaces()


   def make_text(self,message):
      font = pg.font.SysFont('Arial', self.size)
      text = font.render(message,True,self.color)
      rect = text.get_rect(center = (self.srect.
      centerx, self.srect.centery + self.buff_centery) )
      return text,rect

   def make_surfaces(self):
      self.text = []
      for i, line in enumerate(self.lst):
         l = self.make_text(line)
         l[1].y += i*self.buff_lines
         self.text.append(l)

   def update(self):
      if pg.time.get_ticks()-self.timer > self.delay:
         self.timer = pg.time.get_ticks()
         for text, rect in self.text:
            rect.y -= 1

   def render(self, surf):
      for text, rect in self.text:
         surf.blit(text, rect)

screen = pg.display.set_mode((800,600))
screen_rect = screen.get_rect()
clock = pg.time.Clock()
running=True
cred = Credits(screen_rect, text_list)


while running:
   for event in pg.event.get():
      if event.type == QUIT:
         running = False
   screen.fill((0,0,0))
   cred.update()
   cred.render(screen)
   pg.display.update()
   clock.tick(60)
```

A Long Time Ago...

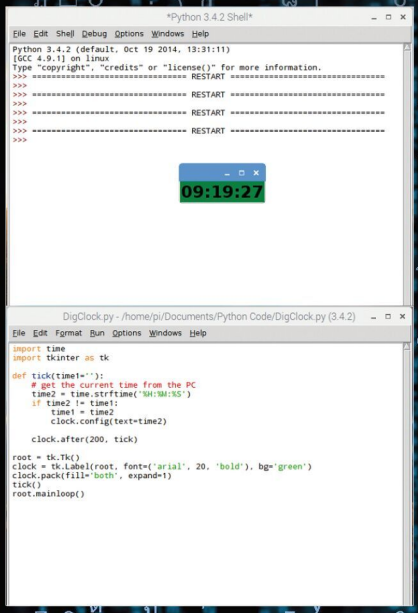
The obvious main point of enhancement is the actual text itself. Replace it with a list of credits, or an equally epic opening storyline to your Python game, and it will certainly hit the mark with whoever plays it. Don't forget to change the screen resolution if needed; we're currently running it at 800 x 600.





Python Digital Clock

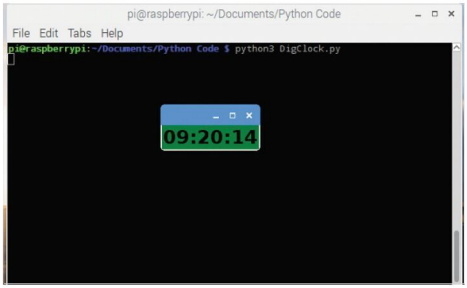
There is already a clock displayed on the desktop of most operating systems but it's always handy to have one on top of the currently open window. To that end, why not create a Python digital clock that can be a companion desktop widget for you.

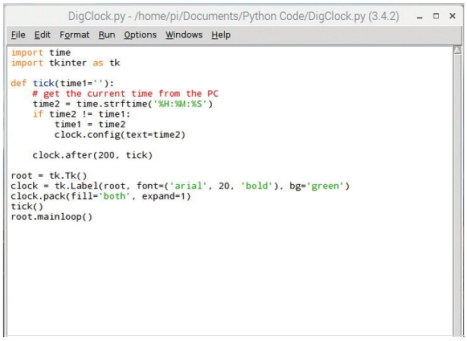


DIGCLOCK.PY

This is a surprisingly handy little script and one that we've used in the past instead of relying on a watch or even the clock in the system tray of the operating system.

```
import time
import tkinter as tk

def tick(time1=''):
   # get the current time from the PC
   time2 = time.strftime('%H:%M:%S')
   if time2 != time1:
      time1 = time2
      clock.config(text=time2)

   clock.after(200, tick)

root = tk.Tk()
clock = tk.Label(root, font=('arial', 20, 'bold'),
bg='green')
clock.pack(fill='both', expand=1)
tick()
root.mainloop()
```



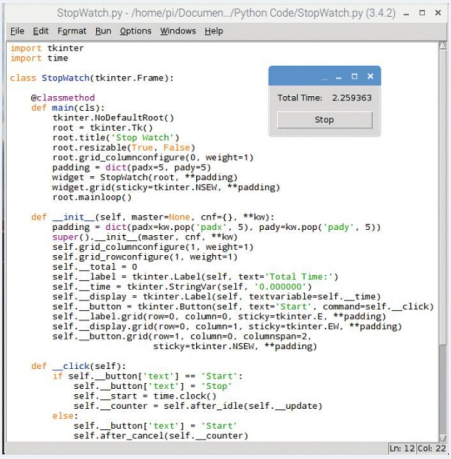


Tick Tock

This is a piece of code we've used many times in the past to keep track of time while working on multiple monitors and with just a quick glance to where we've placed it on the screen.

The Tkinter box can be moved around without affecting the time, maximised or closed by the user at will. We haven't given the Tkinter clock window a title, so you can add to that easily enough by snipping the code from other examples in this book.

Another area of improvement is to include this code when Windows or Linux starts, so it automatically pops up on the desktop. See also, if you're able to improve its functionality by including different time zones: Rome, Paris, London, New York, Moscow and so on.



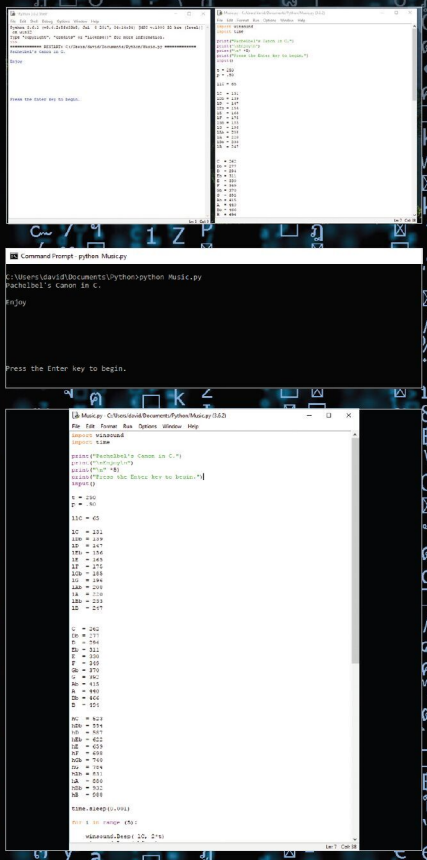
Another example, expanding on the original code, could be a digital stopwatch. For that you could use the following:

```
import tkinter
import time

class StopWatch(tkinter.Frame):

    @classmethod
    def main(cls):
        tkinter.NoDefaultRoot()
        root = tkinter.Tk()
        root.title('Stop Watch')
        root.resizable(True, False)
        root.grid_columnconfigure(0, weight=1)
        padding = dict(padx=5, pady=5)
        widget = StopWatch(root, **padding)
        widget.grid(sticky=tkinter.NSEW, **padding)
        root.mainloop()

    def __init__(self, master=None, cnf={}, **kw):
        padding = dict(padx=kw.pop('padx', 5), pady=kw.pop('pady', 5))
        super().__init__(master, cnf, **kw)
        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(1, weight=1)
        self.__total = 0
        self.__label = tkinter.Label(self, text='Total Time:')
        self.__time = tkinter.StringVar(self, '0.000000')
        self.__display = tkinter.Label(self, textvariable=self.__time)
        self.__button = tkinter.Button(self, text='Start', command=self.__click)
        self.__label.grid(row=0, column=0, sticky=tkinter.E, **padding)
        self.__display.grid(row=0, column=1, sticky=tkinter.EW, **padding)
        self.__button.grid(row=1, column=0, columnspan=2,
                           sticky=tkinter.NSEW, **padding)

    def __click(self):
        if self.__button['text'] == 'Start':
            self.__button['text'] = 'Stop'
            self.__start = time.clock()
            self.__counter = self.after_idle(self.__update)
        else:
            self.__button['text'] = 'Start'
            self.after_cancel(self.__counter)
```

```
        root.title('Stop Watch')
        root.resizable(True, False)
        root.grid_columnconfigure(0, weight=1)
        padding = dict(padx=5, pady=5)
        widget = StopWatch(root, **padding)
        widget.grid(sticky=tkinter.NSEW, **padding)
        root.mainloop()

    def __init__(self, master=None, cnf={}, **kw):
        padding = dict(padx=kw.pop('padx', 5), pady=kw.
        pop('pady', 5))
        super().__init__(master, cnf, **kw)
        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(1, weight=1)
        self.__total = 0
        self.__label = tkinter.Label(self,
        text='Total Time:')
        self.__time = tkinter.StringVar(self,
            '0.000000')
        self.__display = tkinter.Label(self,
        textvariable=self.__time)
        self.__button = tkinter.Button(self,
        text='Start', command=self.__click)
        self.__label.grid(row=0, column=0,
        sticky=tkinter.E, **padding)
        self.__display.grid(row=0, column=1,
        sticky=tkinter.EW, **padding)
        self.__button.grid(row=1, column=0,
        columnspan=2,sticky=tkinter.NSEW, **padding)

    def __click(self):
        if self.__button['text'] == 'Start':
            self.__button['text'] = 'Stop'
            self.__start = time.clock()
            self.__counter = self.after_idle(self.__update)
    else:
        self.__button['text'] = 'Start'
        self.after_cancel(self.__counter)

    def __update(self):
        now = time.clock()
        diff = now - self.__start
        self.__start = now
        self.__total += diff
        self.__time.set('{:.6f}'.format(self.__total))
        self.__counter = self.after_idle(self.__update)

if __name__ == '__main__':
    StopWatch.main()
```

Playing Music with the Winsound Module

Of course, instead of playing an existing MP3, you can always make your own music. The code below will play out Pachelbel's Canon in D, no less.



MUSIC.PY

The code utilises both the Time and Winsound modules, defining the tone and pitch and inserting small pauses of .5 of a second.

```
import winsound
import time

t = 250
p = .50

llC = 65

lC  = 131
lDb = 139
lD  = 147
lEb = 156
lE  = 165
lF  = 175
lGb = 185
lG  = 196
lAb = 208
lA  = 220
lBb = 233
lB  = 247

C  = 262
Db = 277
D  = 294
Eb = 311
E  = 330
F  = 349
Gb = 370
G  = 392
Ab = 415
A  = 440
Bb = 466
B  = 494

hC  = 523
hDb = 554
hD  = 587
hEb = 622
hE  = 659
hF  = 698
hGb = 740
hG  = 784
hAb = 831
hA  = 880
hBb = 932
hB  = 988

time.sleep(0.001)
```

1

2

```
for i in range (5):

   winsound.Beep( lC, 2*t)
   winsound.Beep( hC, t)
   winsound.Beep( hE, t)
   winsound.Beep( hG, t)
   time.sleep(p)

   winsound.Beep( lG, 2*t)
   winsound.Beep( G, t)
   winsound.Beep( B, t)
   winsound.Beep( hD, t)
   time.sleep(p)

   winsound.Beep( lA, 2*t)
   winsound.Beep( A, t)
   winsound.Beep( hC, t)
   winsound.Beep( hE, t)
   time.sleep(p)

   winsound.Beep( lE, 2*t)
   winsound.Beep( E, t)
   winsound.Beep( G, t)
   winsound.Beep( B, t)
   time.sleep(p)

   winsound.Beep( lF, 2*t)
   winsound.Beep( F, t)
   winsound.Beep( A, t)
   winsound.Beep( hC, t)
   time.sleep(p)

   winsound.Beep( llC, 2*t)
   winsound.Beep( C, t)
   winsound.Beep( E, t)
   winsound.Beep( G, t)
   time.sleep(p)

   winsound.Beep( lF, 2*t)
   winsound.Beep( F, t)
   winsound.Beep( A, t)
   winsound.Beep( hC, t)
   time.sleep(p)

   winsound.Beep( lG, 2*t)
   winsound.Beep( G, t)
   winsound.Beep( B, t)
   winsound.Beep( hD, t)
   time.sleep(p)
```

3

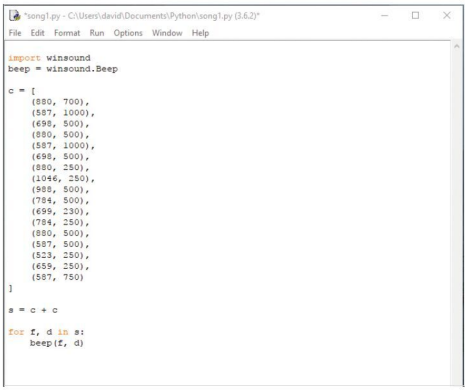
Sweet Music

Obviously the Winsound module is a Windows-only set of functions for Python. Open your IDLE in Windows and copy the code in. Press F5 to save and execute, then press the Enter key, as instructed in the code, to start the music.

Naturally you can swap out the winsound.Beep frequency and durations to suit your own particular music; or you can leave it as is and enjoy. Perhaps play around with the various methods to make other music.

For example, players of the Nintendo classic game, The Legend of Zelda: Ocarina of Time, can enjoy the game's titular musical intro by entering:

```
import winsound
beep = winsound.Beep

c = [
    (880, 700),
    (587, 1000),
    (698, 500),
    (880, 500),
    (587, 1000),
    (698, 500),
    (880, 250),
    (1046, 250),
    (988, 500),
    (784, 500),
    (699, 230),
    (784, 250),
    (880, 500),
    (587, 500),
    (523, 250),
    (659, 250),
    (587, 750)
]

s = c + c

for f, d in s:
    beep(f, d)
```



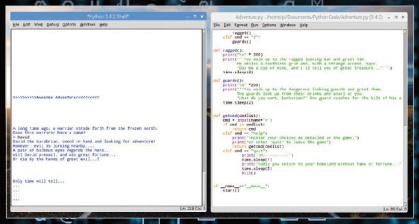
1 The start of the code imports the Winsound and Tie modules; remember, this is a Windows-only Python script. The variable t is setting the duration, while p equals .5, which you can use for the time.sleep function.

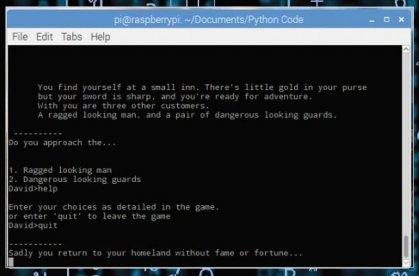
2 These variables set the frequencies, with the corresponding numbers, which can be used in the next section of the code.

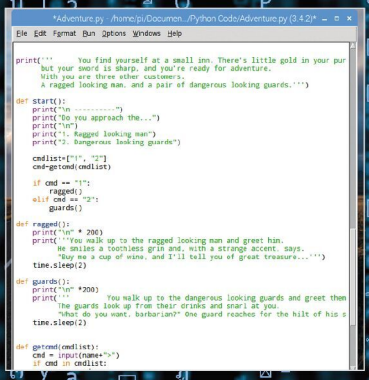
3 Winsound.beep requires a frequency and duration within the brackets. The frequencies come from the large set of variables called in the second section of the code and the duration is through the t variable set at the start of the code. There's a half-second, using the variable p, pause between blocks of winsound.beep statements.

Text Adventure Script

Text adventures are an excellent way to build your Python coding skills and have some fun at the same time. This example that we created will start you on the path to making a classic text adventure; where it will end is up to you.







ADVENTURE.PY

The Adventure game uses just the Time module to begin with, creating pauses between print functions. There's a help system in place to expand upon, as well as the story itself.

```
import time

print("\n" * 200)
print(">>>>>>>>>>Awesome Adventure<<<<<<<<<<\n")
print("\n" * 3)
time.sleep(3)
print("\nA long time ago, a warrior strode forth from
the frozen north.")
time.sleep(1)
print("Does this warrior have a name?")
name=input("> ")
print(name, "the barbarian, sword in hand and looking
for adventure!")
time.sleep(1)
print("However, evil is lurking nearby....")
time.sleep(1)
print("A pair of bulbous eyes regards the hero...")
time.sleep(1)
print("Will", name, "prevail, and win great fortune...")
time.sleep(1)
print("Or die by the hands of great evil...?")
time.sleep(1)
print("\n" *3)
print("Only time will tell...")
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(5)
print("\n" *200)

print('''       You find yourself at a small inn. There's
   little gold in your purse but your sword is sharp,
   and you're ready for adventure.
   With you are three other customers.
   A ragged looking man, and a pair of dangerous
   looking guards.''')

def start():
   print("\n ----------")
   print("Do you approach the...")
   print("\n")
   print("1. Ragged looking man")
   print("2. Dangerous looking guards")

   cmdlist=["1", "2"]
   cmd=getcmd(cmdlist)
```

```
    if cmd == "1":
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and
    greet him.
        He smiles a toothless grin and, with a strange
        accent, says.
        "Buy me a cup of wine, and I'll tell you of
        great treasure...'''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''You walk up to the dangerous looking guards
    and greet them.
        The guards look up from their drinks and
        snarl at you.
        "What do you want, barbarian?" One guard reaches
        for the hilt of his sword...'''')
    time.sleep(2)
```

```
def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
        return cmd
    elif cmd == "help":
        print("\nEnter your choices as detailed in
        the game.")
        print("or enter 'quit' to leave the game")
        return getcmd(cmdlist)
    elif cmd == "quit":
        print("\n----------")
        time.sleep(1)
        print("Sadly you return to your homeland without
        fame or fortune...")
        time.sleep(5)
        exit()


if __name__=="__main__":
    start()
```

Adventure Time

This, as you can see, is just the beginning of the adventure and takes up a fair few lines of code. When you expand it, and weave the story along, you'll find that you can repeat certain instances such as a chance meeting with an enemy or the like.

We've created each of the two encounters as a defined set of functions, along with a list of possible choices under the cmdlist list, and cmd variable, of which is also a defined function. Expanding on this is quite easy, just map out each encounter and choice and create a defined function around it. Providing the user doesn't enter quit into the adventure, they can keep playing.

There's also room in the adventure for a set of variables designed for combat, luck, health, endurance and even an inventory or amount of gold earned. Each successful combat situation can reduce the main character's health but increase their combat skills or endurance. Plus, they could loot the body and gain gold, or earn gold through quests.

Finally, how about introducing the Random module. This will enable you to include an element of chance in the game. For example, in combat, when you strike an enemy you will do a random amount of damage as will they. You could even work out the maths behind improving the chance of a better hit based on your or your opponent's combat skills, current health, strength and endurance. You could create a game of dice in the inn, to see if you win or lose gold (again, improve the chances of winning by working out your luck factor into the equation).

Needless to say, your text adventure can grow exponentially and prove to be a work of wonder. Good luck, and have fun with your adventure.

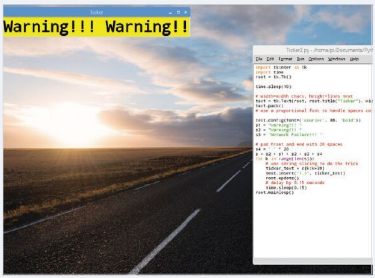
Python Scrolling Ticker Script

You may be surprised to hear that one of the snippets of code we're often asked for is some form of scrolling ticker. Whilst we've covered various forms of scrolling text previously, the ticker is something that seems to keep cropping up. So, here it is.

Ticker Time

The obvious improvements to the Ticker code lie in the speed of the text and what the text will display. Otherwise you can change the background colour of the ticker window, the font and the font colour, along with the geometry of the Tkinter window if you want to.

Yet another interesting element that could be introduced is one of the many text to Speech modules available for Python 3. You could pip install one, import it, then as the ticker displays the text, the text to speech function will read out the variable at the same time, since the entire text is stored in the variable labelled 's'.



The ticker example can be used for system warnings, perhaps something that will display across your work or home network detailing the shutting down of a server over the weekend for maintenance; or even just to inform everyone as to what's happening. We're sure you will come up with some good uses for it.

TICKER.PY

We're using Tkinter here along with the Time module to determine the speed the text is displayed across the window.

```
import time
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, root.title("Ticker Code"),
height=80, width=600, bg="yellow")
canvas.pack()
font = ('courier', 48, 'bold')
text _ width = 15

#Text blocks insert here....

s1 = "This is a scrolling ticker example. As you
can see, it's quite long but can be a lot longer if
necessary... "
s2 = "We can even extend the length of the ticker
message by including more variables... "
s3 = "The variables are within the s-values in
the code. "
s4 = "Don't forget to concatenate them all before the
For loop, and rename the 'spacer' s-variable too."

# pad front and end of text with spaces
s5 = ' ' * text _ width
# concatenate it all
s = s5 + s1 + s2 + s3 + s4 + s5
x = 1
y = 2
text = canvas.create _ text(x, y, anchor='nw', text=s,
font=font)
dx = 1
dy = 0  # use horizontal movement only

# the pixel value depends on dx, font and length of text
pixels = 9000

for p in range(pixels):
   # move text object by increments dx, dy
   # -dx --> right to left
   canvas.move(text, -dx, dy)
   canvas.update()
   # shorter delay --> faster movement
   time.sleep(0.005)
   #print(k)  # test, helps with pixel value

root.mainloop()
```

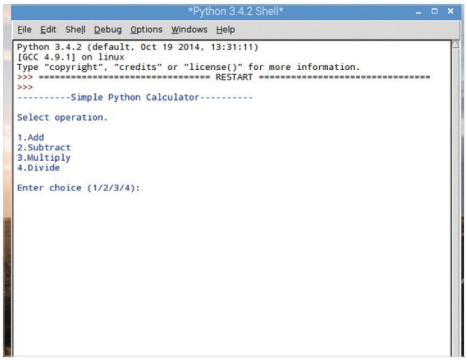
Simple Python Calculator

Sometimes the simplest code can be the most effective. Take for example, this Simple Python Calculator script. It's based on the Create Your Own Modules section seen earlier but doesn't utilise any external modules.

CALCULATOR.PY

We created some function definitions to begin with, then lead on to the user menu and inputs. It's an easy piece of code to follow and as such can also be expanded well too.

```
print("----------Simple Python Calculator----------\n")

def add(x, y):
   return x + y

def subtract(x, y):
   return x - y

def multiply(x, y):
   return x * y

def divide(x, y):
   return x / y

print("Select operation.\n")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter first number: "))
num2 = int(input("Enter second number: "))

if choice == '1':
   print(num1,"+",num2,"=", add(num1,num2))

elif choice == '2':
   print(num1,"-",num2,"=", subtract(num1,num2))

elif choice == '3':
   print(num1,"*",num2,"=", multiply(num1,num2))

elif choice == '4':
   print(num1,"/",num2,"=", divide(num1,num2))
else:
   print("Invalid input")
```



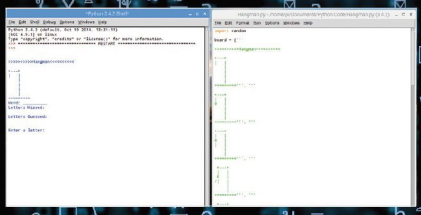
Improved Calculations

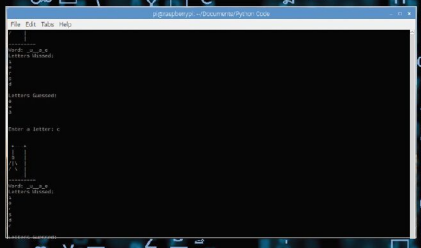
The obvious contender for improvement here is using the Create Your Own Modules route and extracting the function definitions as a module. You can then call the module and focus on the body of the code.

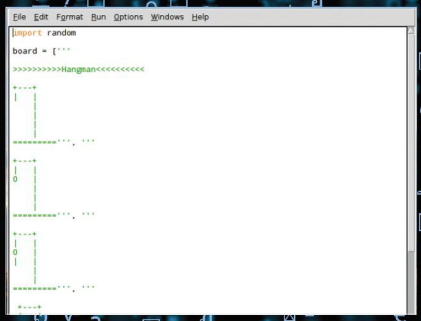
The other area of improvement is code itself. Where there's just a single shot at making a calculation, you could encase it in a while loop, so once a value is presented the user is sent back to the main menu. Perhaps, improvement to the Invalid Input section is worth looking into as well.

Hangman Game Script

Hangman is a great game to program into Python. It can be extremely complex, displaying graphics, the number of guesses left in the secret word, a huge bank of available words picked at random and countless other elements. It can also be quite simple. Here we have a mix between the two.







HANGMAN.PY

We've made a Hangman game board (the gallows) out of characters that can be displayed in the IDLE Shell, along with a huge bank of words to randomly choose from.

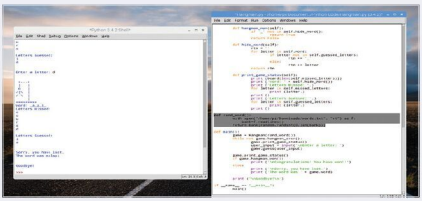
```
import random

board = ['''

>>>>>>>>>>Hangman<<<<<<<<<<

+---+
|   |
    |
    |
    |
    |
=========''', '''

+---+
|   |
O   |
    |
    |
    |
=========''', '''

+---+
|   |
O   |
|   |
    |
    |
=========''', '''

 +---+
 |   |
 O   |
/|   |
     |
     |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
     |
     |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
/    |
```

```
        |
=========''', '''

 +---+
 |   |
 O   |
/|\  |
/ \  |
     |
=========''']


class Hangman:
   def __init__(self,word):
      self.word = word
      self.missed_letters = []
      self.guessed_letters = []

   def guess(self,letter):
      if letter in self.word and letter not in self.
      guessed_letters:
         self.guessed_letters.append(letter)
      elif letter not in self.word and letter not in
      self.missed_letters:
         self.missed_letters.append(letter)
      else:
         return False
      return True

   def hangman_over(self):
      return self.hangman_won() or (len(self.missed_
      letters) == 6)

   def hangman_won(self):
      if '_' not in self.hide_word():
         return True
      return False

   def hide_word(self):
      rtn = ''
      for letter in self.word:
         if letter not in self.guessed_letters:
            rtn += '_'
         else:
            rtn += letter
      return rtn

   def print_game_status(self):
      print (board[len(self.missed_letters)])
      print ('Word: ' + self.hide_word())
      print ('Letters Missed: ',)
      for letter in self.missed_letters:
         print (letter,)
      print ()
      print ('Letters Guessed: ',)
      for letter in self.guessed_letters:
         print (letter,)
      print ()

def rand_word():
   bank = 'ability about above absolute accessible
   accommodation accounting beautiful bookstore
   calculator clever engaged engineer enough
   handsome refrigerator opposite socks interested
   strawberry backgammon anniversary confused
   dangerous entertainment exhausted impossible
   overweight temperature vacation scissors
   accommodation appointment decrease development
   earthquake environment brand environment necessary
   luggage responsible ambassador circumstance
   congratulate frequent'.split()
   return bank[random.randint(0,len(bank))]

def main():
   game = Hangman(rand_word())
   while not game.hangman_over():
      game.print_game_status()
      user_input = input('\nEnter a letter: ')
      game.guess(user_input)

   game.print_game_status()
   if game.hangman_won():
      print ('\nCongratulations! You have won!!')
   else:
      print ('\nSorry, you have lost.')
      print ('The word was ' + game.word)

   print ('\nGoodbye!\n')

if __name__ == "__main__":
   main()
```

QUIT()

Since this is the last example in our Python code repository, we thought we'd go out with a bang and feature the hangman gallows being drawn with each incorrect guess of the word. Don't worry if it looks misaligned in the text here, this is merely due to the differences between using the Python IDLE editor and pasting the code into a word processor (which formats things differently).

There's plenty you can do to improve, enhance and expand on what we've presented here. You can include a routine that returns an error if the user enters a number or character. You can include extra points for someone who guesses the entire word in one go rather than one letter at a time and you could perhaps add Chopin's Funeral March should you lose the game; or something celebratory if you win.



Consider replacing the bank of words too. They're found under the bank list, and could easily be swapped out for something more difficult. If you download www.github.com/dwyl/english-words you can find a text document with over 466,000 words. Perhaps you could swap the words in the bank to instead read the contents of the text file:

```
def rand_word():
   with open("/home/pi/Downloads/words.txt", "rt") as f:
      bank=f.readlines()
   return bank[random.randint(0,len(bank))]
```

Understanding Linux

Linux is a remarkably versatile and powerful operating system. It's used throughout the programming and engineering world, in science, space exploration, education, gaming and everything else in between. It's the OS of choice for high-performance servers, it's the backbone of the Internet and it powers the fastest supercomputers in the world.

Knowing how to use Linux, and how it's structured, is key to being able to create better Python content. The Raspberry Pi, for example, uses a Linux-based OS and, as such, makes for an excellent coding platform. Regardless of whether you're using a Pi, like us, or a Linux Mint or Ubuntu, these pages will prove invaluable for your Python learning. Master Linux, master Python, and start engineering your coding future.

What is Linux?

The Raspberry Pi operating system is Raspbian, which is a Linux operating system; but what exactly is Linux? Where did it come from and what does it do? In a world where Windows and macOS have supremacy of the desktop, it's easy to overlook it, but there's more to Linux than you might imagine.

Linux is a surprisingly powerful, fast, secure and capable operating system. It's used as the OS of choice for the Raspberry Pi, in the form of Raspbian OS, as well as in some of the most unlikely places.

Despite only enjoying a 1.96% share (according to netmarketshare.com) of the total desktop operating system market, Linux has a dedicated following of enthusiasts, users and contributors. It was created in 1991 by University of Helsinki student, Linus Torvalds, who had become frustrated with the limitations and licensing of the popular educational system Minix, a miniature version of the Unix operating system, in use at the time.

Unix itself was released in the early '70s, as a multi-tasking, modular-designed operating system originally developed for programmers who needed a stable platform to code on. However, its performance, power and portability meant that it soon became the system of choice for companies and universities where high-end computing tasks were needed.

Torvalds needed a system that could mirror Unix's performance and features, without the licensing cost. Thus was born Linux, the Unix-like operating system which used freely available code from the GNU project. This enabled users around the world to utilise the power of the Unix-like system, completely free of charge, an ethos that still holds today: Linux is free to download, install and use.

Linux is much like any other operating system, such as Windows or macOS in that it manages the computer hardware, provides an interface for the user to access that hardware and comes with programs for productivity, communications, gaming, science, education and more. Linux can be broken up into a number of significant elements:

BOOTLOADER

The bootloader is the software that initialises and boots up your computer. It loads up the various modules the OS uses to begin to access the hardware in the system. You can modify a bootloader to load more than one OS installed on the system.

DAEMONS

Daemons are background services that start as the operating system is booting. These can enable printing, sound, networking and so on. They run unobtrusively rather than under the direct control of the user, often waiting to be activated by an event or condition.

DESKTOP ENVIRONMENTAL

The Desktop Environment, or DE, is the main Graphical User Interface (GUI) that users interact with. It's the desktop, that includes Internet browsers, productivity, games and whatever program or app you're using. There are countless DEs available. Raspbian uses PIXEL.

GRAPHICAL SERVER

This is a module within Linux that provides a graphical output to your monitor. It's referred to as the X server or simply just X. X is an application that manages one or more graphical displays and one or more input devices (keyboard, mouse, etc.) connected to the computer.

KERNEL

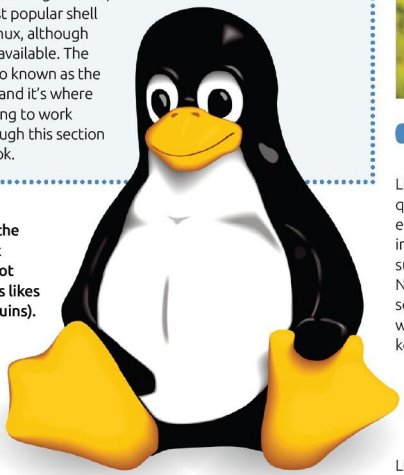
The kernel is the core of the system and the single element that is actually called Linux. The Linux kernel manages the computer processor, memory, storage and any peripherals you have attached to your computer. It provides the basic services for all other parts of the OS.

PROGRAMS/APPLICATIONS

With Linux being an open source, free operating system, it also makes use of the tens of thousands of freely available applications. The likes of LibreOffice, GIMP and Python are just the tip of the iceberg.

SHELL

The Linux shell is a command-line interface environment that a Linux user can use to enter commands to the OS that directly affect it. Within the shell you can add new users, reboot the system, create and delete files and folders, and much more. BASH (Bourne Again Shell) is the most popular shell used in Linux, although more are available. The shell is also known as the Terminal, and it's where you're going to work from through this section of the book.



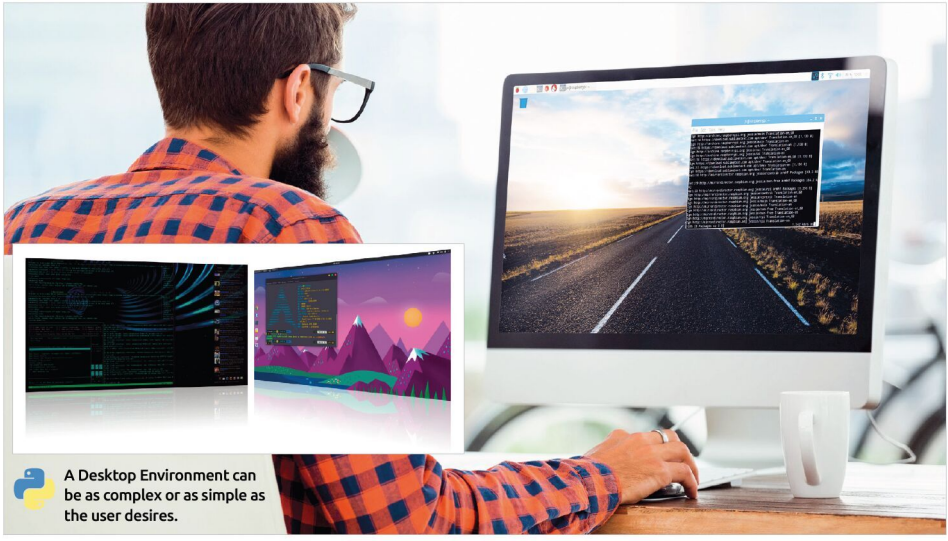
Tux, the Linux mascot (Linus likes penguins).

Linus Torvalds, the creator of the Linux kernel.

Linux is used throughout the world, in a number of basic and quite unique uses. While it may look radically different from one environment to the next, the actual Linux kernel, can be found in modern smart TVs, in-car entertainment systems and GPS, supercomputers, IoT devices and the Raspberry Pi. It's used by NASA, both in the command centre and on-board the ISS. Linux servers power the backbone of the Internet, along with most of the websites you visit daily. Android utilises components of the Linux kernel, as do set top boxes, games consoles and even your fridge, freezer, oven and washing machine.

Linux isn't just a free to use operating system. It's stable, powerful and fast, easily customised and requires very little maintenance. However, it's more than just performance stats; Linux means freedom from the walled garden approach of other operating systems. It's a lively community of like-minded individuals who want more from their computers without the shackles of price or conformity. Linux means choice.

Raspbian on the Raspberry Pi, is the Linux distribution of choice.



A Desktop Environment can be as complex or as simple as the user desires.

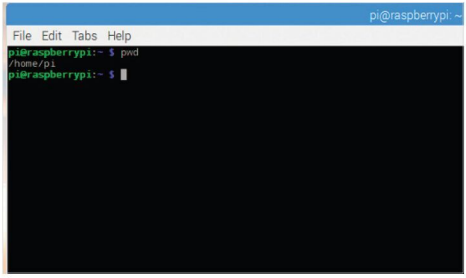
Using the Filesystem

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.

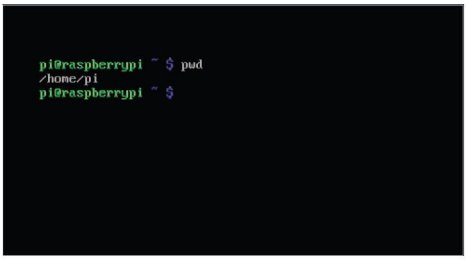
GETTING AROUND

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.



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.

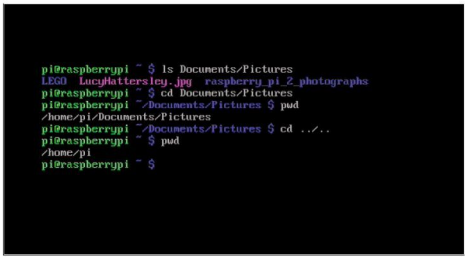


STEP 3 Enter: `ls` to view the contents of the current directory. You should see Desktop, Documents, and Downloads and Scratch in Blue. You may also see other items depending on how much you have used your Raspberry Pi. The colour code is worth knowing: directories are blue while most files are white. 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.

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.

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.

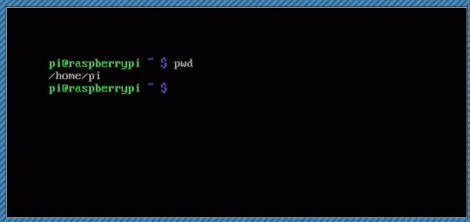
STEP 6 The "ls" and "cd" commands can also be used with more complex paths. Enter: `ls Documents/Pictures` to view the contents of a Pictures directory inside your Documents directory. You can switch to this directory using `cd Documents/Pictures`; use `cd ../..` to move back up two parent directories.



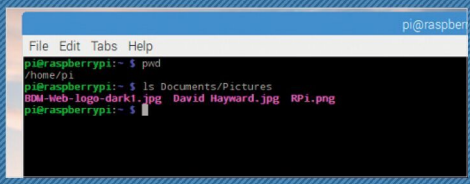
ABSOLUTE VS RELATIVE PATHS

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...

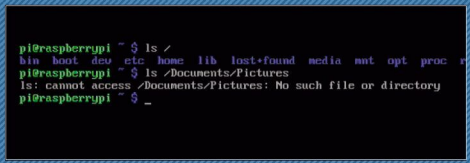
STEP 1 By default, commands like "ls" use the working directory. This is the current directory that you're looking at and is set to your home directory by default (/users/pi). Using "pwd" (Print Working Directory) lets you know what the working directory is, and using "cd" changes the working directory.



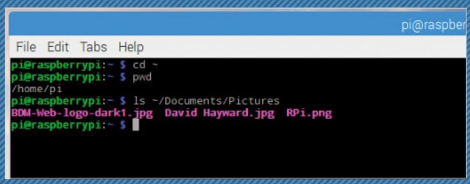
STEP 2 The root directory is always '/'. Entering: `ls /` lists the contents of root, and entering: `cd /` switches to the root directory. This is important because there is a difference between "ls Documents/Pictures" and "ls /Documents/Pictures". The first command lists the contents of the Pictures directory in Documents inside the working directory (which, if you are in the home directory, will work).



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.



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.



Listing and Moving Files

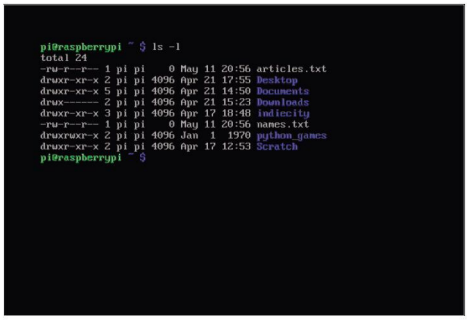
Admittedly, using the desktop GUI to list and move files is much easier than using the Terminal and keyboard. However, it's an important skill that you will appreciate as you advance with the Raspberry Pi and Linux.

LOOKING AT FILES

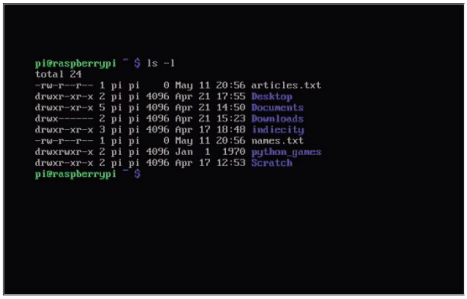
Operating systems are built on files and folders, or directories if you prefer. While you're used to viewing your own files, most operating systems keep other files out of sight. In Raspbian, you have access to every file in the system.

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 '-lah') is known as the argument. This is an option that modifies the behaviour of the command.

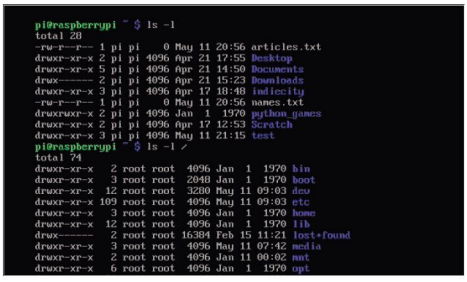
STEP 2 The "-l" argument lists files and directories in long format. Each file and directory is now on a single line, and before each file is a lot of text. First you'll see lots of letters and dashes, like 'drwxr-xr-x'. Don't worry about these for now; they are known as 'permissions' and we'll come to those later.



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.



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.



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 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   37 May 11 21:27 names.txt
drwxrwxr-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 -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
```

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 '"s –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-------  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 .dmrc
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 10:15 .fontconfig
```

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 Raspbian 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 11 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
```

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  gze
bunzip2   bzgrep      chmod     date          dumpkeys    fgrep      gzi
bzcat     bzip2       chown     dd            echo        findmnt    hos
bzcmp     bzip2recover chvt     df            ed          fuser      ip
bzdiff    bzless      con2fbmap dir           egrep       fusermount kbd
bzegrep   bzmore      cp        dmesg         false       grep       kil
bzexe     cat         cpio      dnsdomainname fbset       gunzip     kma
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 find you find program tools and libraries.

```
pi@raspberrypi ~ $ ls
articles.txt  Desktop  Documents  Downloads  indiecity  names.txt  python_g
pi@raspberrypi ~ $ ls /home/pi
articles.txt  Desktop  Documents  Downloads  indiecity  names.txt  python_g
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  ww
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 Raspbian 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
drwx------ 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
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 11:15 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 full stop in place of a space though; stick to underscores.

```
pi@raspberrypi ~ $ touch don't.use(odd)symbols&in<filenames>or=you'll^confu
```

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
drwx------ 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
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 11:15 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
drwx------ 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
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 11:15 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
drwxrwxr-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
drwxrwxr-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 aren't as easy in the Terminal as they are 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.

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".

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.

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.

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_ga
pi@raspberrypi ~ $ rmdir testdir3_
```

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 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_ga
pi@raspberrypi ~ $ rm -R articles
pi@raspberrypi ~ $ ls
Desktop  Documents  Downloads  indiecity  python_games  Scrat
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

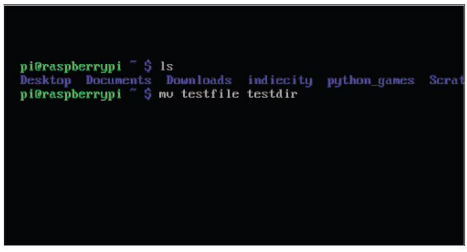
Taking command of the Terminal is essential when learning how your Raspberry Pi's operating system works. The copying, moving and renaming of files is equally important, as you'll be doing a lot of this throughout your Pi projects.

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.

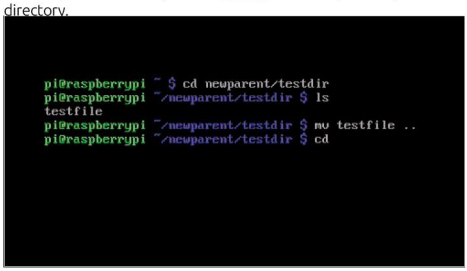
STEP 2 Files and directories are moved using the mv command. This is different to the commands we've looked at so far because it has two arguments (remember Linux command line is command, option, argument). The first argument is the source (the file or directory to be moved) and the second is the destination.



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.

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.

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.

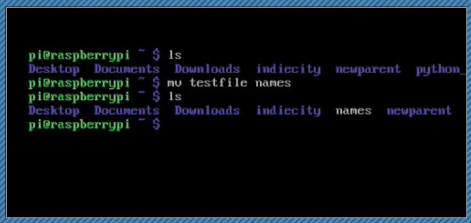


STEP 6 You can also move files using longer paths. Enter: `cd ~` to return to the home directory and `mv newparent/testfile newparent/testdir/testfile` to move the testfile from its current location back inside the testdir directory. Enter: `ls newparent/testdir` to view the file back in its current directory.

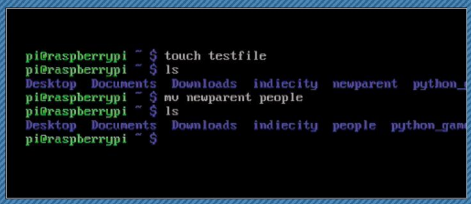
RENAMING FILES AND DIRECTORIES

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.

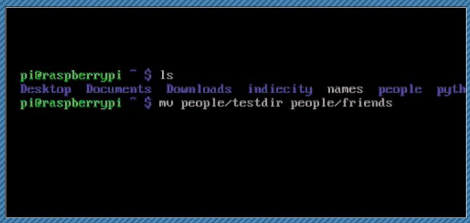
STEP 1 Let's start by making a new test file called "names". Enter: `touch testfile` and then `ls` to make sure the testfile is present. We're going to turn this into a file that contains the names of some people. So let's call it something more appropriate, like "names".



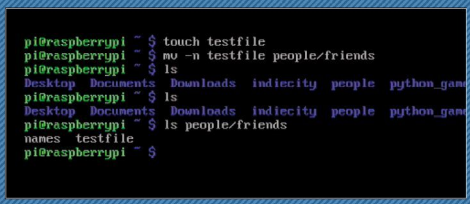
STEP 2 Enter: `mv testfile names` and `ls`. Now we can see the new "names" file in our directory. The mv command can also be used to rename directories. We should still have our newparent directory in our home directory. Enter: `mv newparent people` to rename the newparent directory. Enter: `ls` to view it.



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.



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.



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 evt
CPU implementer : 0x41
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 evt
CPU implementer : 0x41
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 evt
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc07
CPU revision    : 5
```

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:         884304 kB
MemFree:          78576 kB
MemAvailable:     834008 kB
Buffers:          11660 kB
Cached:           56360 kB
SwapCached:           0 kB
Active:           451480 kB
Inactive:         30072 kB
Active(anon):     11256 kB
Inactive(anon):     228 kB
Active(file):     37892 kB
Inactive(file):   29844 kB
Unevictable:          0 kB
Mlocked:              0 kB
SwapTotal:        102396 kB
SwapFree:         102396 kB
Dirty:                36 kB
Writeback:             0 kB
AnonPages:        11132 kB
Mapped:           24164 kB
Shmem:              256 kB
Slab:             16384 kB
SReclaimable:      9512 kB
SUnreclaim:        6872 kB
KernelStack:        672 kB
PageTables:         564 kB
NFS_Unstable:         0 kB
Bounce:               0 kB
WritebackTmp:         0 kB
CommitLimit:      544548 kB
Committed_AS:      45712 kB
VmallocTotal:   1171456 kB
VmallocUsed:       3880 kB
VmallocChunk:    922360 kB
pi@raspberrypi ~ $ cat /proc/partitions
major minor  #blocks  name

 179        0    7761920 mmcblk0
 179        1     835890 mmcblk0p1
 179        2          1 mmcblk0p2
```

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 Raspbian 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 17
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 04d9:1503 Holtek Semiconductor, Inc. Shortboard Lefty
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 showdown -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           6501636 3484164   2740096  56% /
/dev/root        6501636 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 Raspbian 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/autostart
```

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 Raspbian) 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  indiecity  people  python_games  Scratch  testfile
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_Tall.png
8.0K    python_games/princess.png
12K     python_games/Selector.png
8.0K    python_games/4row_black.png
4.0K    python_games/catanimation.py
20K     python_games/flippy.py
36K     python_games/match3.wav
24K     python_games/starpusher.py
```

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

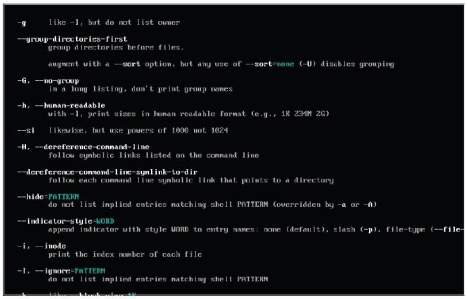
Linux has a built-in manual, known as man for short. Using the man command you can obtain information on all the Linux commands we've talked about. Simply enter: `man` and the name of the command you want to learn more about. Start by entering: `man ls` in the command line.

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.

STEP 3

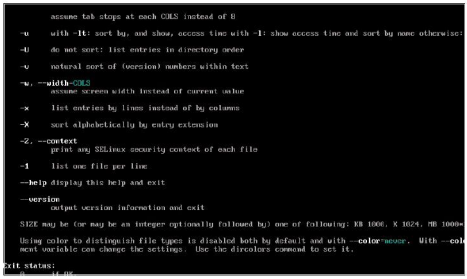
Most commands are pretty easy to figure out how to use, so what you spend most of the time in the man pages is looking under the Description. Here you will see all the options and the letters used to activate them. Most man pages are longer than a single page, so press any key, such as the space bar,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



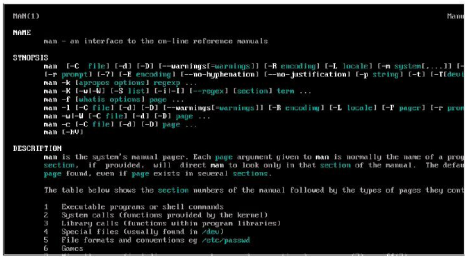
STEP 4

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.

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.



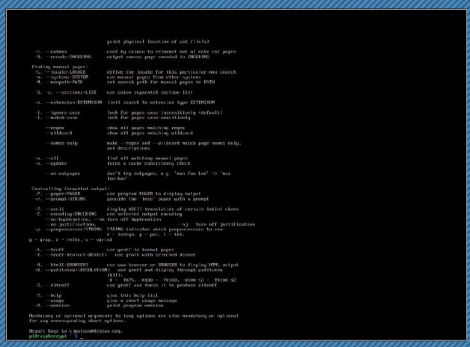
STEP 6 The man command can be used for just about every command you use in Linux. You can even enter: man man to get information on using the man tool. From now on, whenever you come across a new command in this book, such as "nano" or "chmod", take time to enter: `man nano` or `man chmod` and read the instructions.



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.

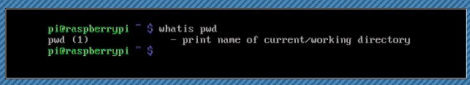
STEP 1 Entering: `man man` enables you to view some of the options, but sometimes you'll just want a quick overview. Fortunately man has a built-in help option that quickly lists the options. Press Q if you're in a man page and enter: `man -h` at the command line.



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.

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.

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").



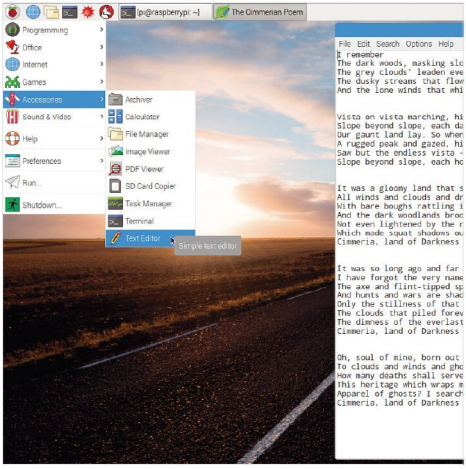
Editing Text Files

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.

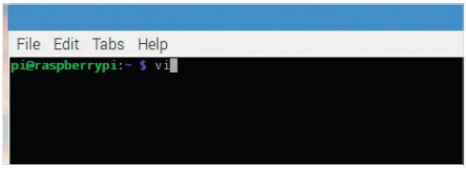
THE JOY OF TEXT

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.

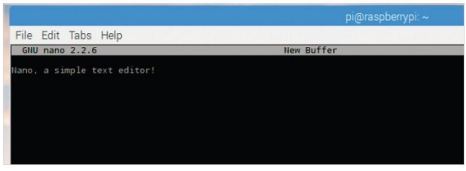


STEP 2 From the Terminal there are even more options, although using the correct command, you can launch any of the desktop apps via the Terminal. One of the simplest, and a classic text editor that's carried over from the Unix days, is vi. In the Terminal, enter: `vi`.

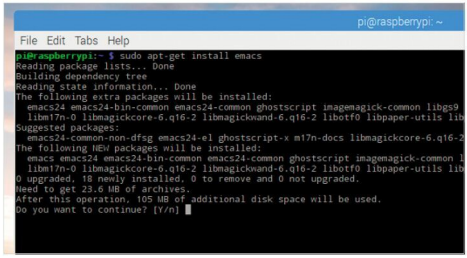


STEP 3 Vi is the original Unix command but in this case it launches VIM, the new Linux version of Vi. Although simple looking, Vi is considered, even by today's standards, to be one of the most widely used text editors, There's a lot you can do with it, so check out the man pages for more Vi information.

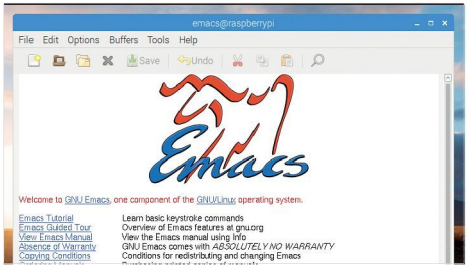
STEP 4 Nano is another favourite, and simple, text editor available for Linux. Enter: `nano` into the Terminal to launch it. You can use Nano for editing code, creating scripts or writing your own help files. To exit Nano, press Ctrl + X, followed by Y to save the file or N to exit without saving.



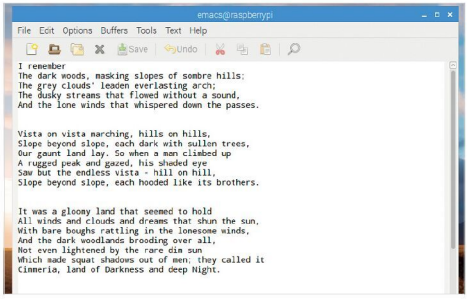
STEP 5 Emacs, or GNU Emacs, is an extensible and customisable, self-documenting, real-time display editor. It's a fantastic text editor and one that's worth getting used to as soon as you can. Sadly, it's not installed on the Pi by default, so you'll need to install it. In the Terminal, enter: `sudo apt-get install emacs`



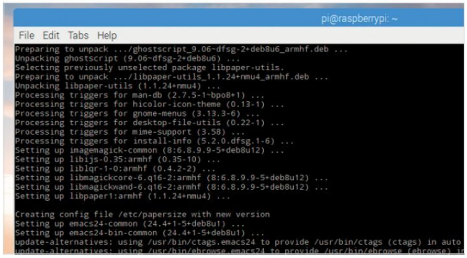
STEP 6 The previous command contacts the Debian (Raspbian is based on a Debian Linux distribution) repositories and pulls down the information needed to install Emacs. When the Pi asks to continue with the installation, press Y. This installs the latest version and when it's done, you'll be back to the command prompt.



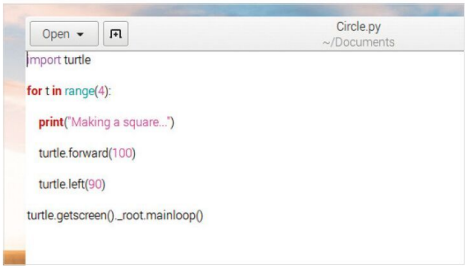
STEP 7 Once the installation is complete, enter: `emacs` into the Terminal. The Emacs splash screen opens in a new window, offering a tutorial (which we recommend you run through) and a guided tour amongst other information.



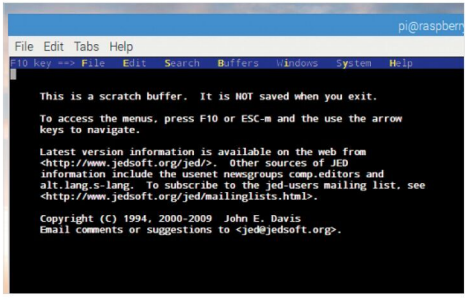
STEP 8 Emacs can offer an uncomplicated view of your text file or one with a plethora of information regarding the structure of the file in question; it's up to you to work out your own preference. There's also a hidden text adventure in Emacs, which we cover later in this book, why not see if you can 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.



Linux Tips and Tricks

The Linux Terminal, you'll no doubt agree, is an exceptional environment and with a few extra apps installed along with a smidgen of command knowledge, incredible and often quite strange things can be accomplished.

TAKING COMMAND

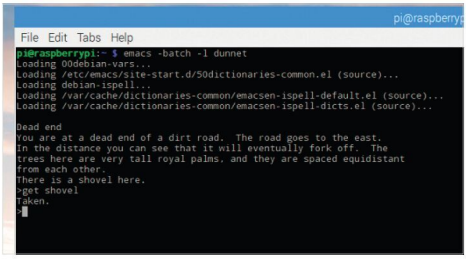
There are countless Linux tips, secrets, hacks and tricks out there. Some are very old, originating from Linux's Unix heritage, while others are recent additions to Linux lore. Here are our ten favourite tips and tricks.

EASTER EGGS

Emacs text editor, is a great piece of software but did you know it also contains a hidden Easter Egg? With Emacs installed (sudo apt-get install emacs24), drop to a Terminal session and enter:

```
emacs -batch -l dunnet
```

Dunnet is a text adventure written by Ron Schnell in 1982, and hidden in Emacs since 1994.



TERMINAL BROWSING

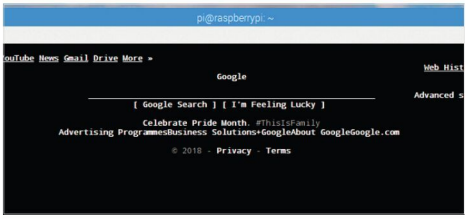
Ever fancied being able to browse the Internet from the Terminal? While not particularly useful, it is a fascinating thing to behold. To do so, enter:

```
sudo apt-get install elinks
```

Then:

```
elinks
```

Enter the website you want to visit.



MOON BUGGY

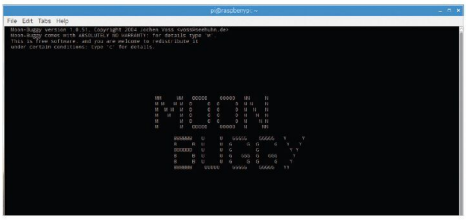
Based on the classic 1982 arcade game, Moon Patrol, Moon Buggy appeared on home computers in 1985 amid much praise. It's a cracking Atari game available in the Linux Terminal by entering:

```
sudo apt-get install moon-buggy
```

Then:

```
moon-buggy
```

Enjoy.



LET IT SNOW

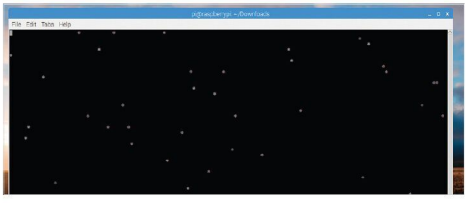
Snowing in the Terminal console isn't something you come across every day. If you're interested, however, enter:

```
wget
```

```
https://gist.githubusercontent.com/sontek/1505483/raw/7d024716ea57e69fb52632fee09f42753361c4a2/snowjob.sh
```

```
chmod +x snowjob.sh
```

```
./snowjob.sh
```

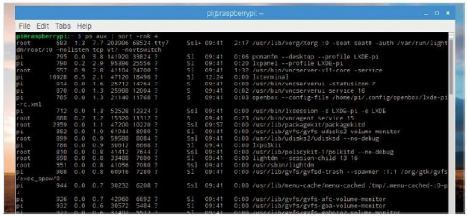


MEMORY HOGS

Memory Hogs – If you need to see which apps are consuming the most memory on your Raspberry Pi, simple enter:

```
ps aux | sort -rnk 4
```

This sorts the output by system memory use.

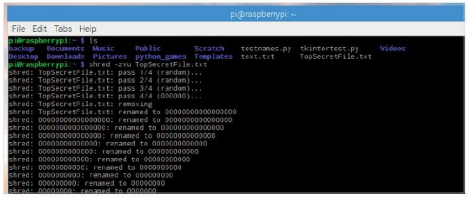


SHREDDER

When you delete a file, there's still a chance of someone with the right software being able to retrieve it. However, files can be securely and permanently deleted using Shred:

```
shred -zvu NAMEOFFILE.txt
```

Replace NAMEOFFILE with the name of the file to delete.

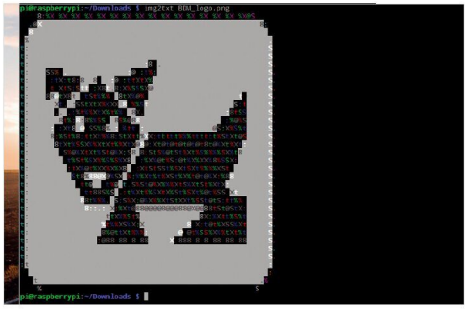


ASCII ART

ASCII art can be quite striking when applied to some images. However, it's often difficult to get just right. You can create some great ASCII art from the images you already have on the Raspberry Pi by using img2txt:

```
img2txt NAMEOFIMAGEFILE.png
```

Replace NAMEOFIMAGEFILE with the actual name of the image file on your system.



BBS

Back in the days of dial-up connections, the online world was made up of Bulletin Board Systems. These remote servers provided hangouts for users to chat, swap code, play games and more. Using Telnet in Linux, you can still connect to some active BBSes:

```
telnet battlestarbbs.dyndns.org
```

There are countless operational BBSes available; check out **www.telnetbbsguide.com/bbs/list/detail/** for more.

DIRECTORY TREES

If you want to create an entire directory (or folder) tree with a single command, you can use:

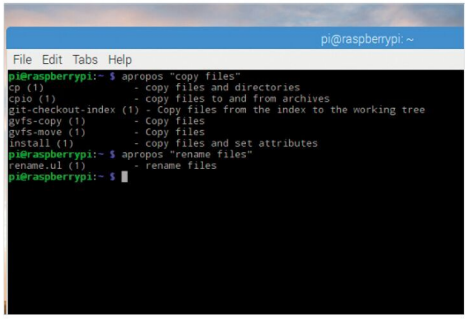
```
mkdir -p New-Dir/
{subfolder1,subfolder2,subfolder3,subfolder4}
```

This creates a New-Dir with four sub folders within.

FORGOTTEN COMMANDS

It's not easy trying to remember all the available Linux commands. Thankfully, you can use apropos to help. Simply use it with a description of the command:

```
apropos "copy files"
apropos "rename files"
```



A-Z of Linux Commands

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

A

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

B

`bc`	An arbitrary precision calculator language

C

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

D

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

E

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

F

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

G

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

H

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

I

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

J

`join`	Join lines on a common field

K

`kill`	Stop a process from running

L

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

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

M

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

N

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

P

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

Q

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

R

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

S

screen	Terminal window manager
sdiff	Merge two files interactively
select	Accept keyboard input
seq	Print numeric sequences
shutdown	Shutdown or restart Linux
sleep	Delay for a specified time
sort	Sort text files
split	Split a file into fixed-size pieces
SSH	Connects to a remote host computer as a specified user, using secure encrypted protocols.
su	Substitute user identity
sudo	Execute a command as another user, primarily as the Root level, administrator user.
sum	Print a checksum for a file
symlink	Make a new name for a file
sync	Synchronise data on disk with memory

T

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

U

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

V

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

W

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

X

xargs	Execute utility, passing constructed argument list(s)

Y

yes	Print a string until interrupted

Glossary of Python Terms

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

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

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

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

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

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

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

Def
Used to define a function or method in Python.

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

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

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

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

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

Float
An immutable floating point number used in Python.

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

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

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

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

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

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

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

Immutable
Something that cannot be changed after it is created.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

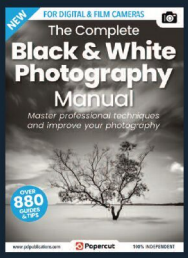
X
Also X11 or X-windows. The graphical desktop used in Linux-based systems, combining visual enhancements and tools to manage the core operating system.

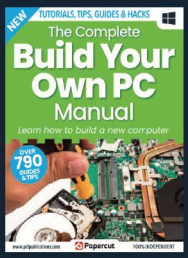
Zen of Python
When you enter: `import this` into the IDLE, the Zen of Python is displayed.

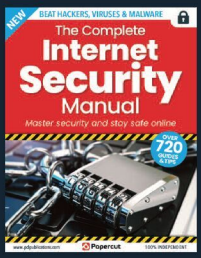
Read More

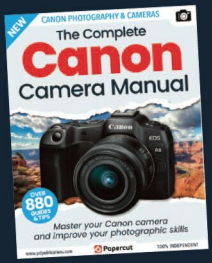
The Complete Manual Series

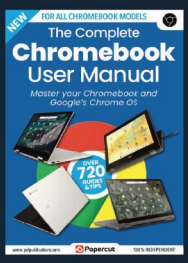
Available on Readly

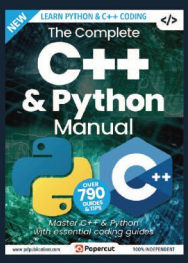


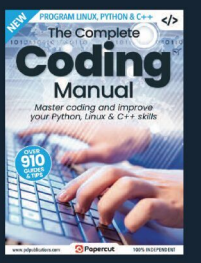


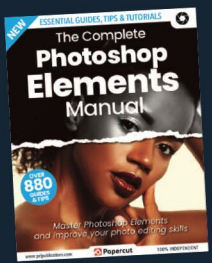


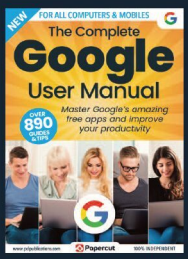


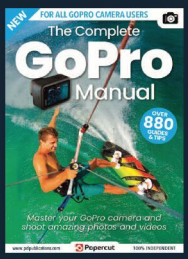


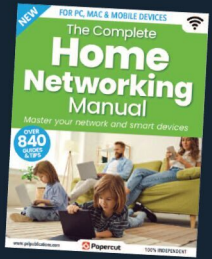


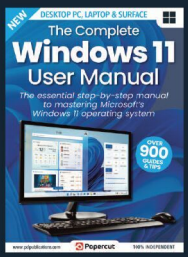


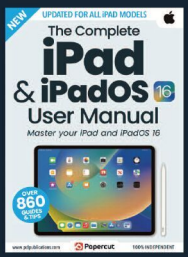












For a full list of titles available visit:

www.pclpublications.com

Congratulations, we have reached the end of your latest tech adventure. With help from our team of tech experts, you have been able to answer all your questions, grow in confidence and ultimately master any issues you had. You can now proudly proclaim that you are getting the absolute best from your latest choice from the ever changing world of consumer technology and software.

So what's next? Do you want to start a new hobby? Are you looking to upgrade to a new device? Or simply looking to learn a new skill?

Whatever your plans we are here to help you. Just check our expansive range of **Tricks & Tips** and **For Beginners** guidebooks and we are positive you will find what you are looking for. This adventure with us may have ended, but that's not to say that your journey is over. Every new hardware or software update brings its new features and challenges, and of course you already know we are here to help. So we will look forward to seeing you again soon.

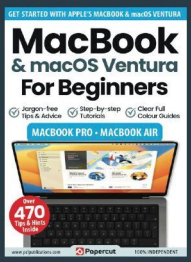
Papercut

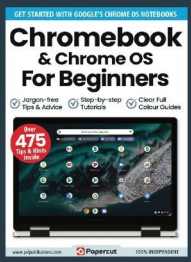
www.pclpublications.com

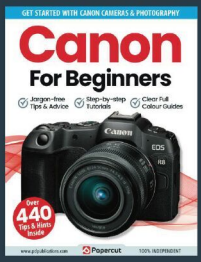
Read More

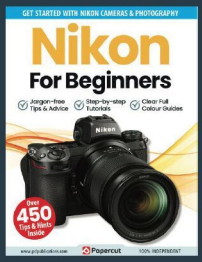
For Beginners

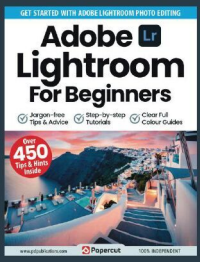
Tech Guides *Available on* Readly

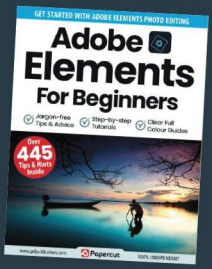


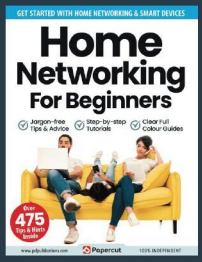


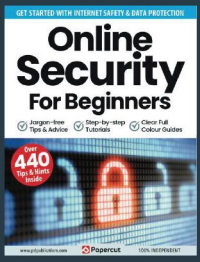


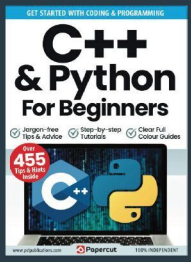


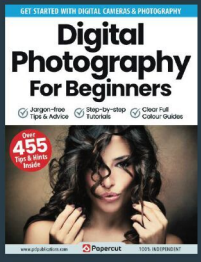


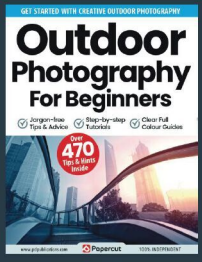


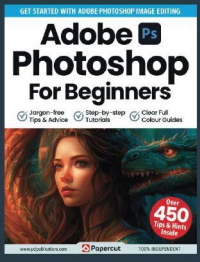


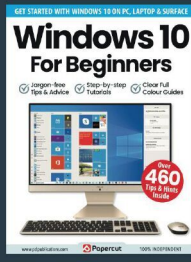












For a full list of titles available visit:

www.pclpublications.com

Python Coding Tricks & Tips
15th Edition | ISBN: 978-1-912847-52-5

Published by: Papercut Limited
Digital distribution by: Readly AB

Papercut Limited
Registered in England & Wales No: 04308513

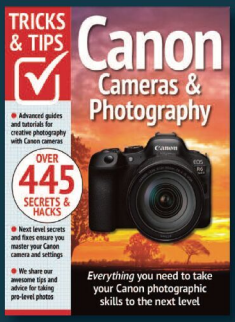
ADVERTISING - For our latest media packs please contact:
Richard Rowe - richard@tandemmedia.co.uk
Will Smith - will@tandemmedia.co.uk

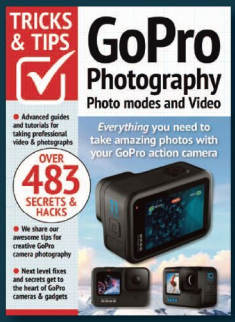
INTERNATIONAL LICENSING - Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: jgale@pclpublications.com

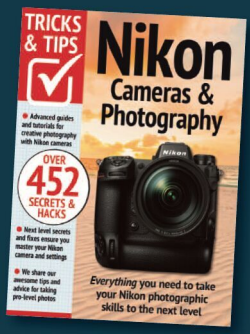
Read More

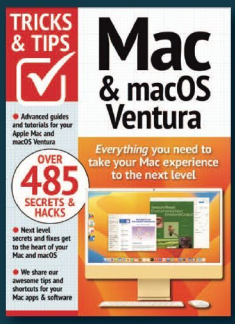
TRICKS & TIPS

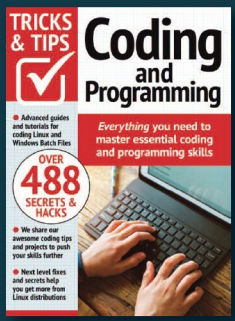
Tech Guides *Available on* Readly

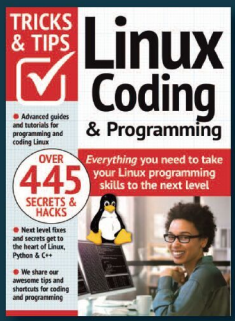












For a full list of titles available visit:

www.pclpublications.com