Using Modules

Modules are where you can take Python programming to new heights. You can create your own modules, or use some of the already available modules, to help convert a mundane piece of code into something spectacular.

Want to see how to make better use of these modules to add a little something extra to your code? Then read on and learn how they can be used to forge fantastic programs.

Calendar Module

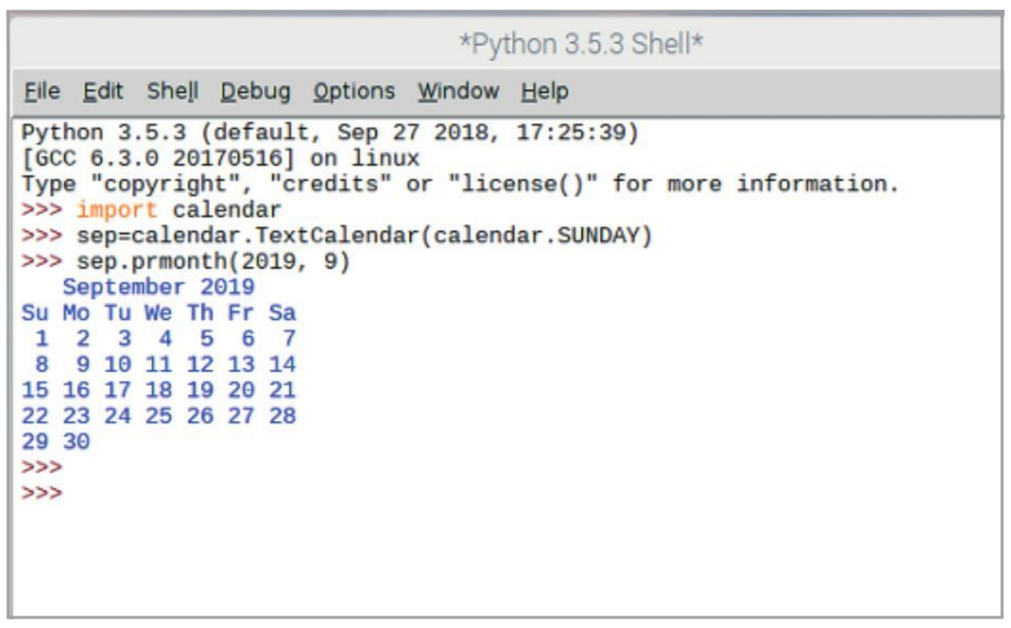
Beyond the Time module, the Calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the Time module-like format, you can actually call up a wall calendar type display.

WORKING WITH DATES

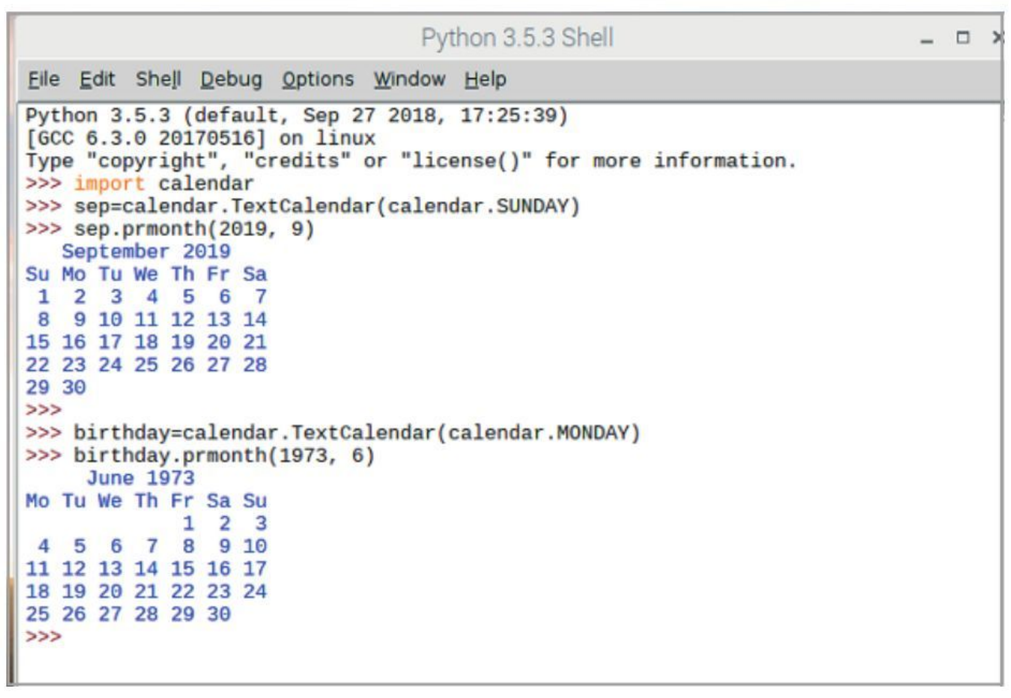
The Calendar module is built into Python 3. However, if for some reason it's not installed you can add it using **pip install calendar** as a Windows administrator, or **sudo pip install calendar** for Linux and macOS.

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)
```



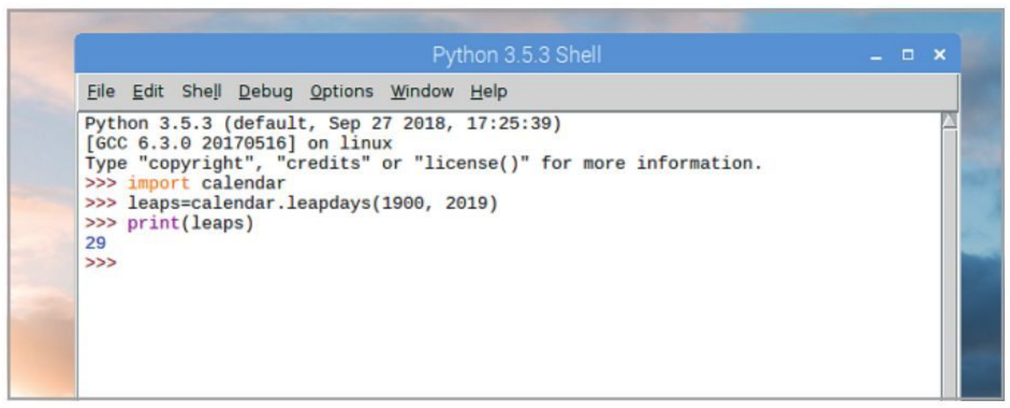
STEP 2 You can see that the days of September 2019 are displayed in a wall calendar fashion. Naturally you can change the 2019, 9 part of the second line to any year and month you want, a birthday for example (1973, 6). The first line configures TextCalendar to start its weeks on a Sunday; you can opt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

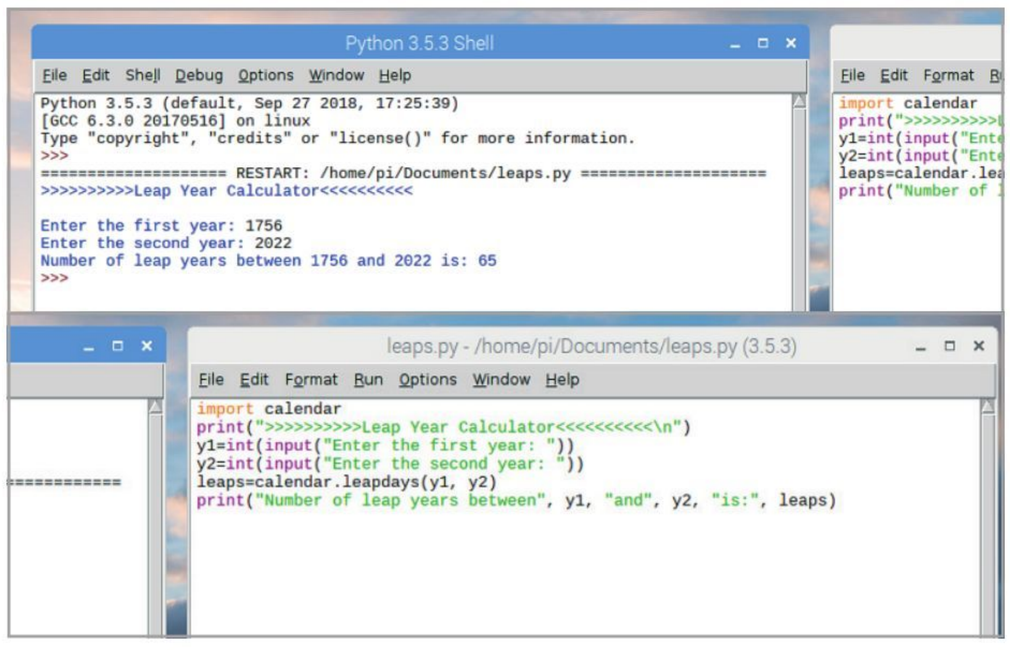
```
leaps=calendar.leapdays(1900, 2019)
print(leaps)
```

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)
```

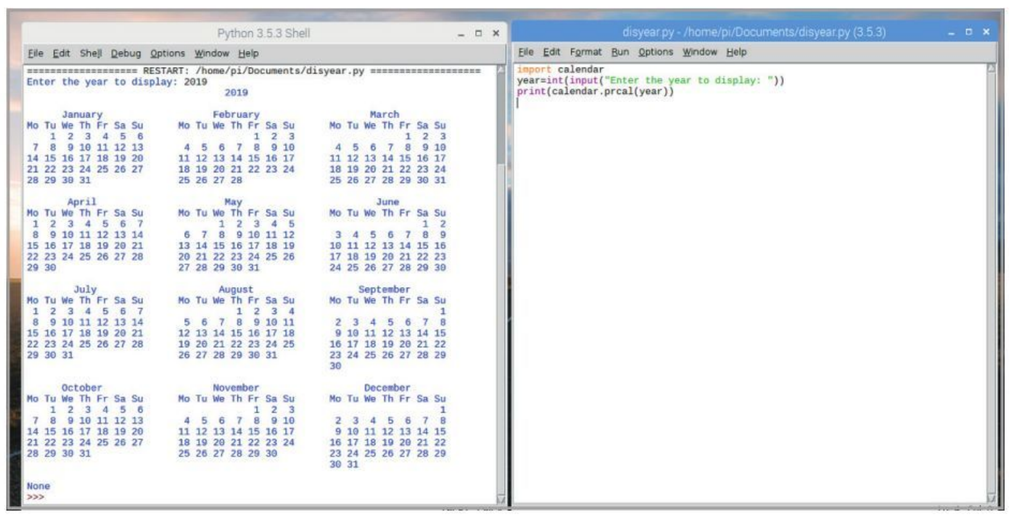


STEP 5

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))
```

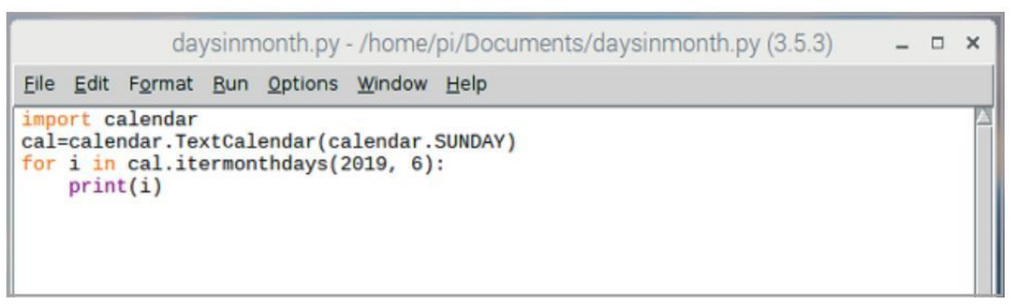
We're sure you'll agree that's quite a handy bit of code to have to hand.



STEP 6

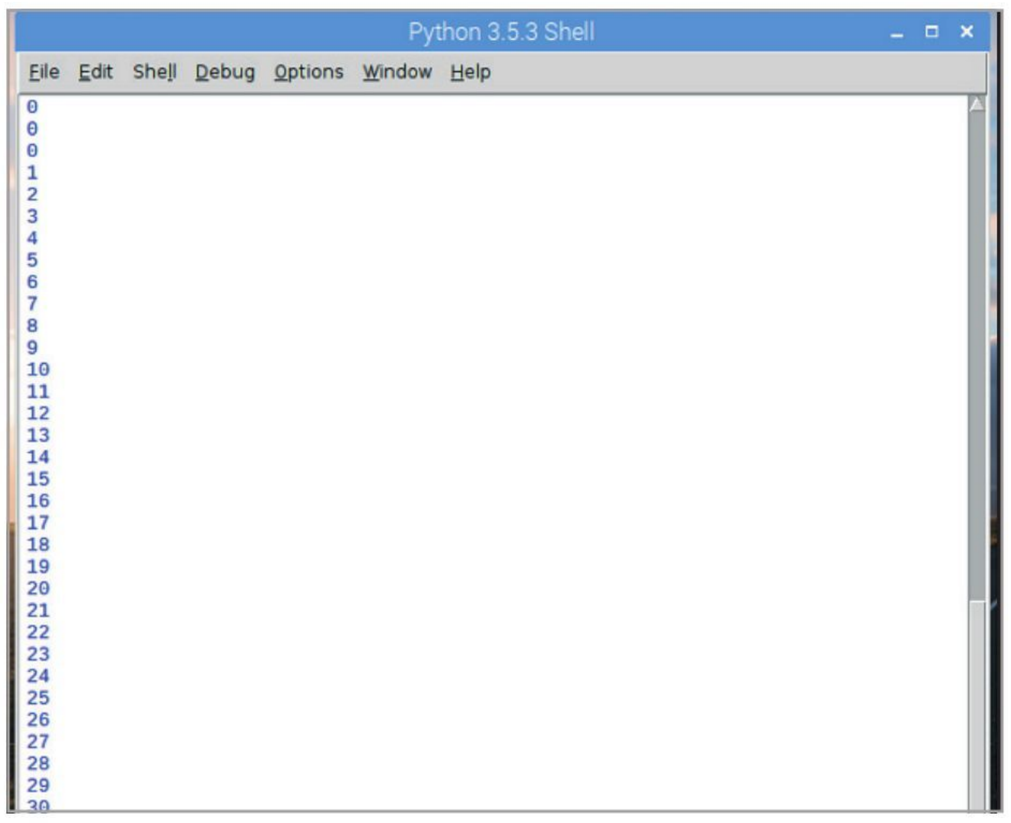
Interestingly we can also list the number of days in a month by using a simple: for loop:

```
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.

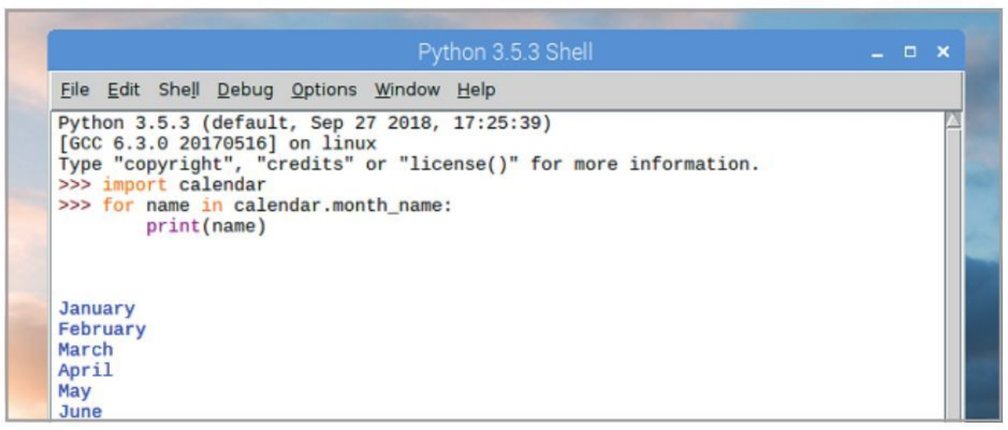


STEP 8

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)
```

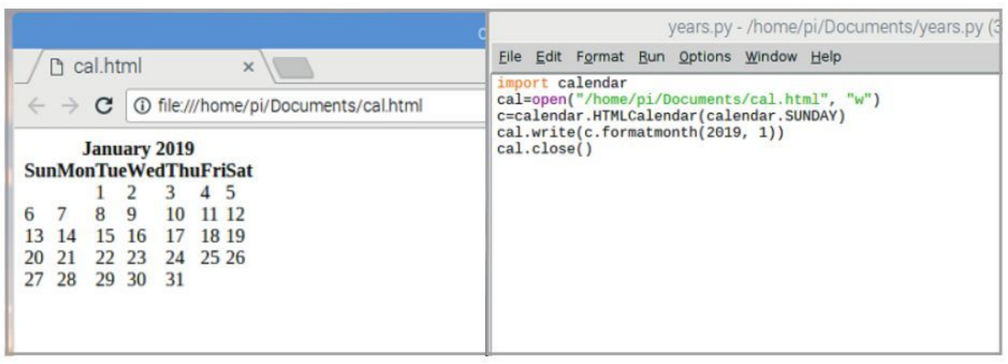


STEP 9

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.



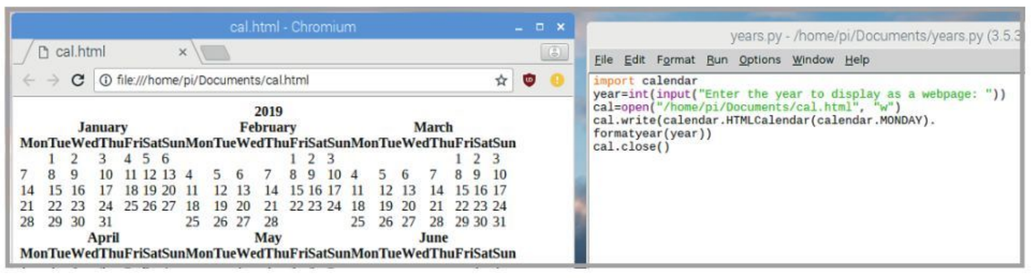
STEP 10

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

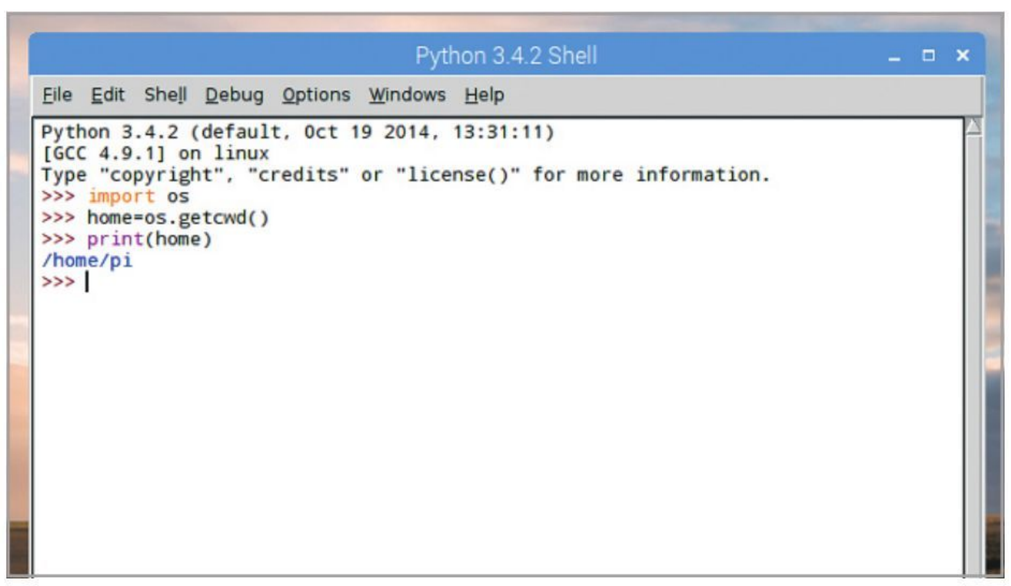
The OS module allows you to interact directly with the built-in commands found in your operating system. Commands vary depending on the OS you're running, as some will work with Windows whereas others will work with Linux and macOS.

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.

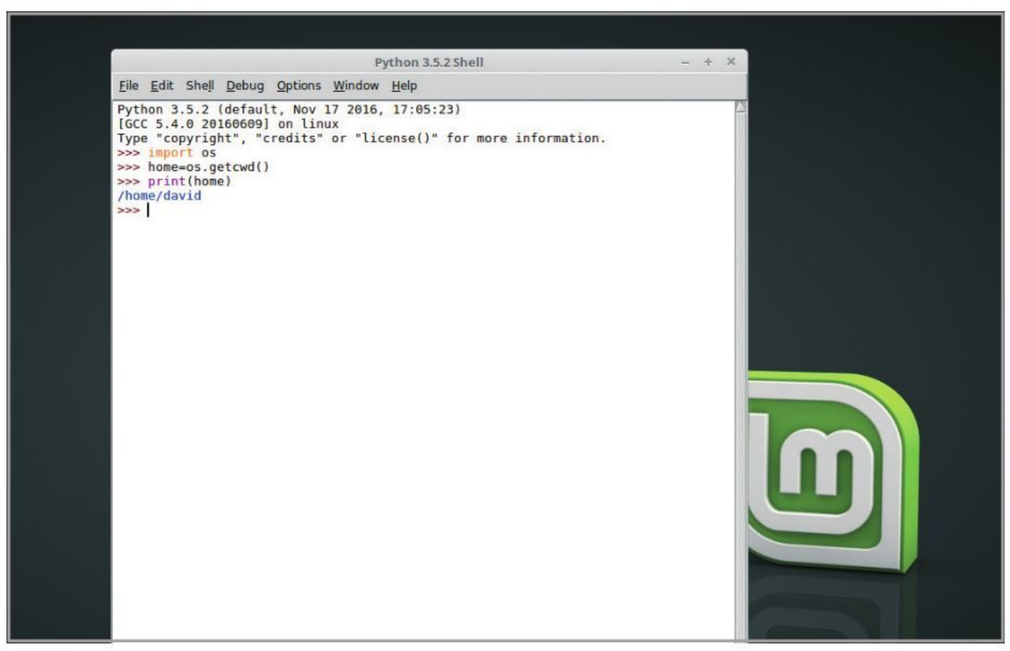
STEP 1 You can start the OS module with some simple functions to see how it interacts with the operating system environment that Python is running on. If you're using Linux or the Raspberry Pi, try this:

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



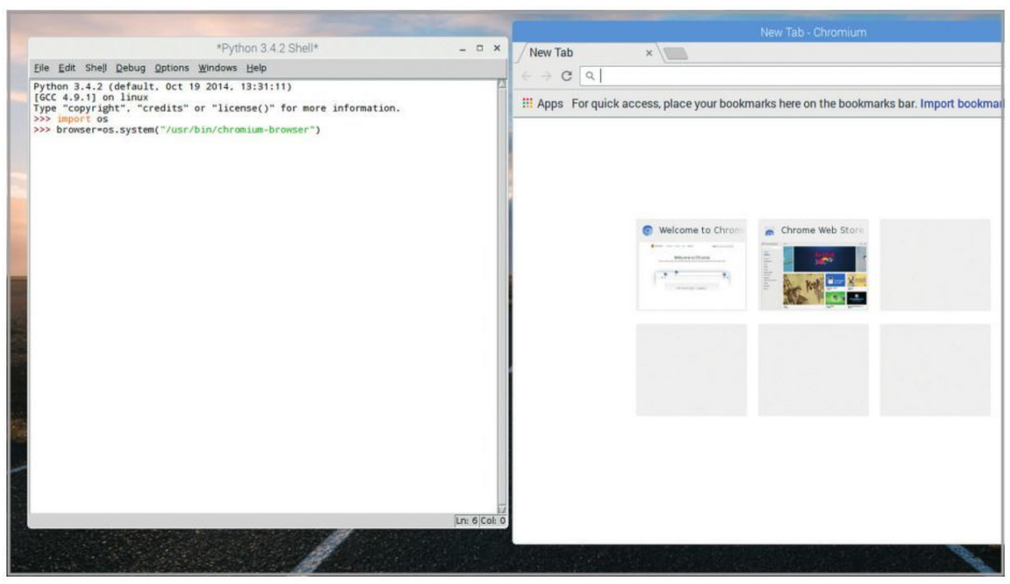
STEP 2 The returned result from printing the variable home is the current user's home folder on the system. In our example that's /home/pi; it will be different depending on the user name you login as and the operating system you use. For example, Windows 10 will output: C:\Program Files (x86)\Python36-32.

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:

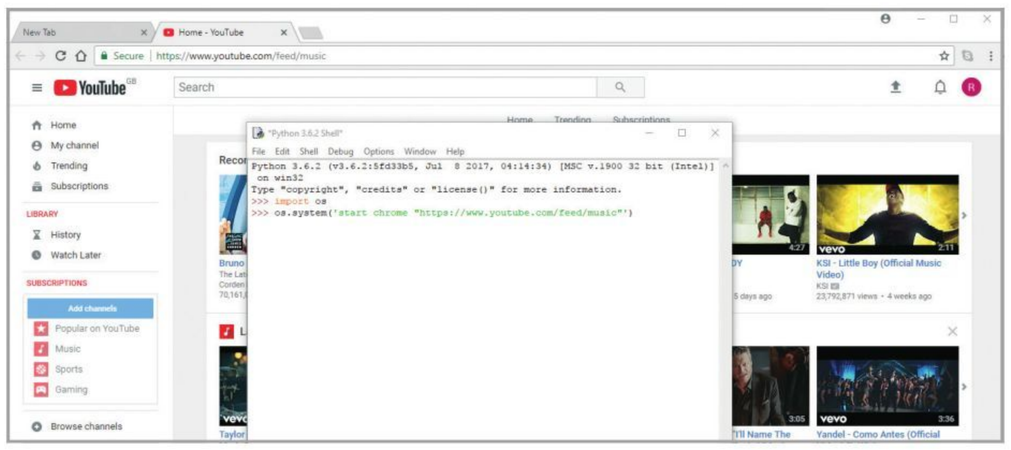
```
import os
browser=os.system("/usr/bin/chromium-browser")
```



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:

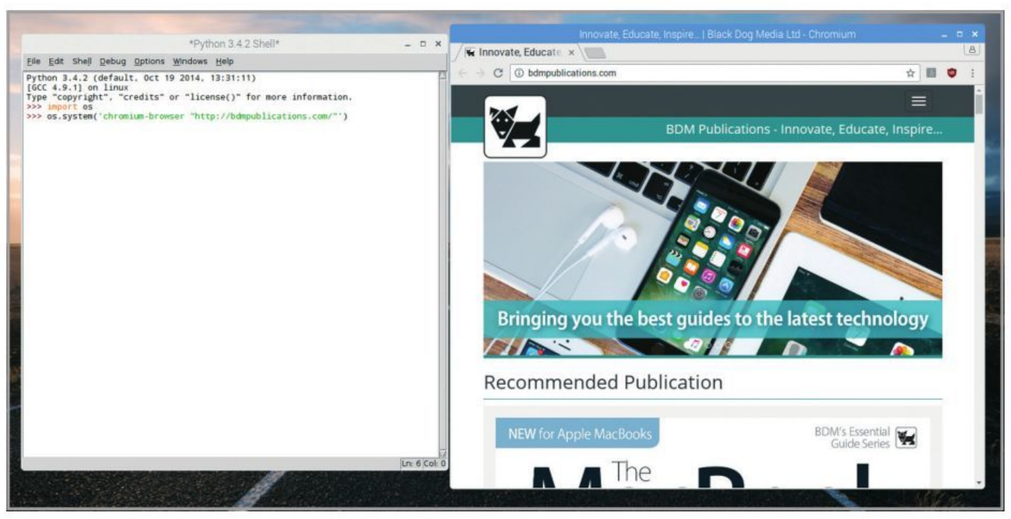
```
import os
os.system('start chrome "https://www.youtube.com/feed/music"')
```



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:

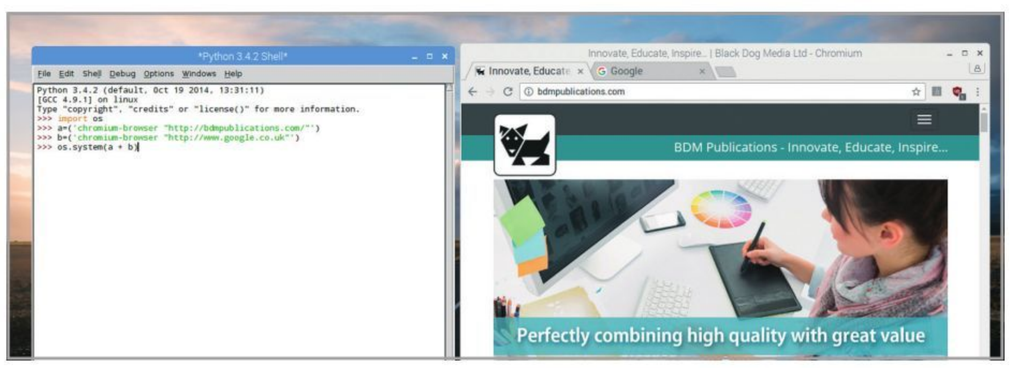
```
import os
os.system('chromium-browser "http://bdmpublications.com/"')
```



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)
```

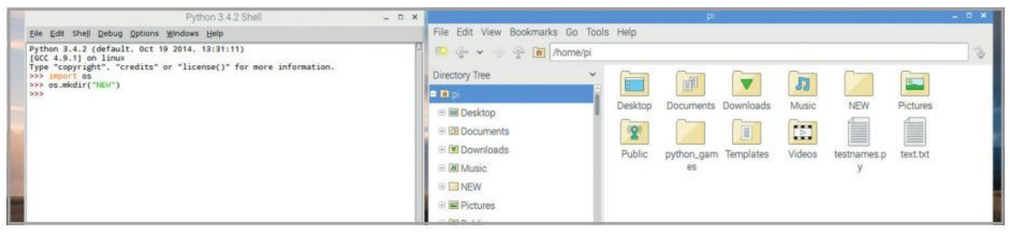


STEP 8

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.



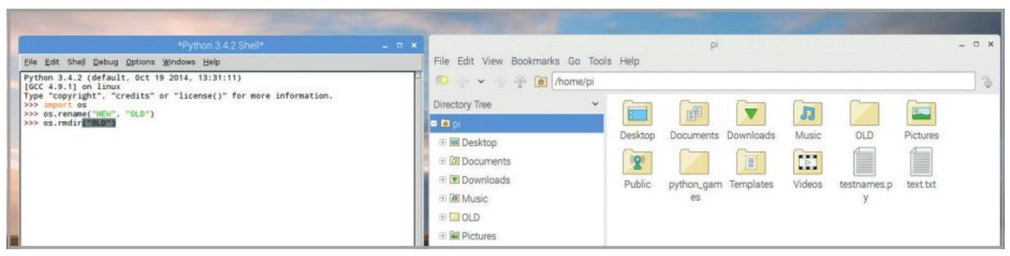
STEP 9

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

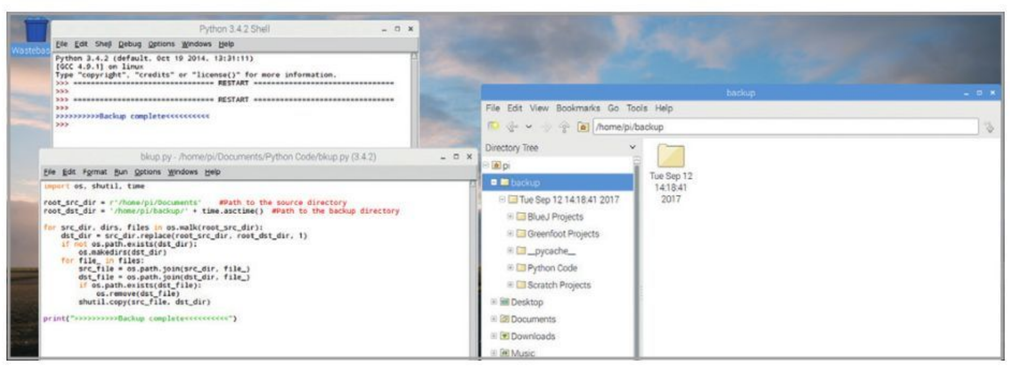
Another module that goes together with OS is shutil. You can use the Shutil module together with OS and time to create a time-stamped backup directory, and copy files into it:

```
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<<<<<<<<<<")
```



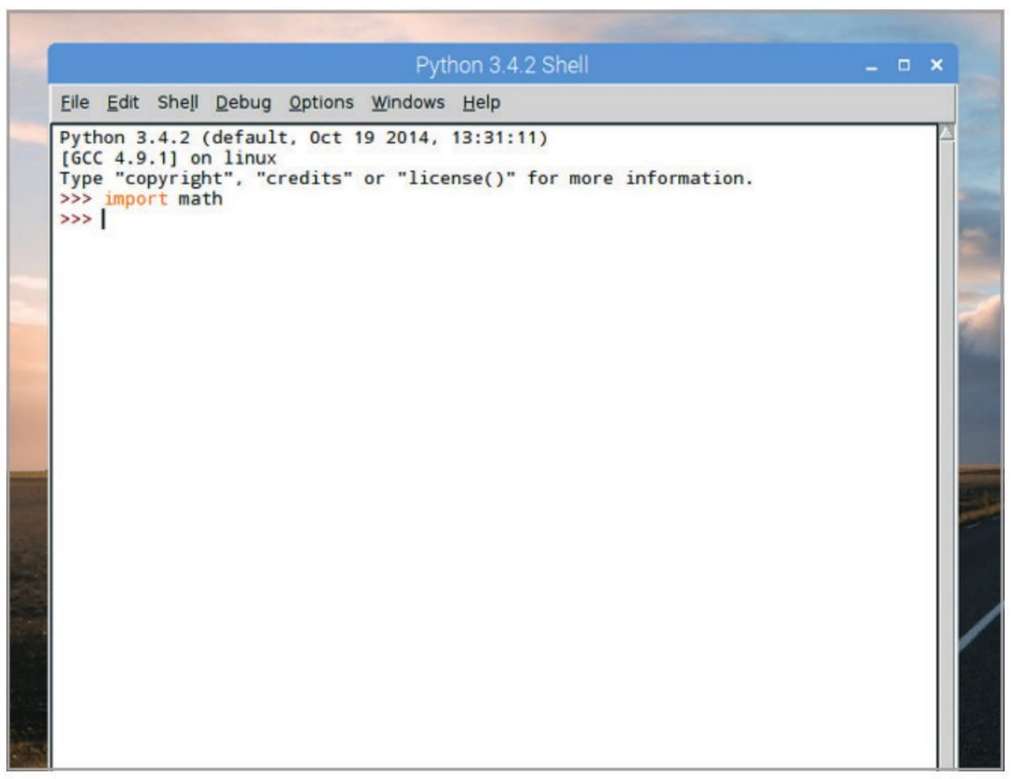
Using the Math Module

One of the more 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.

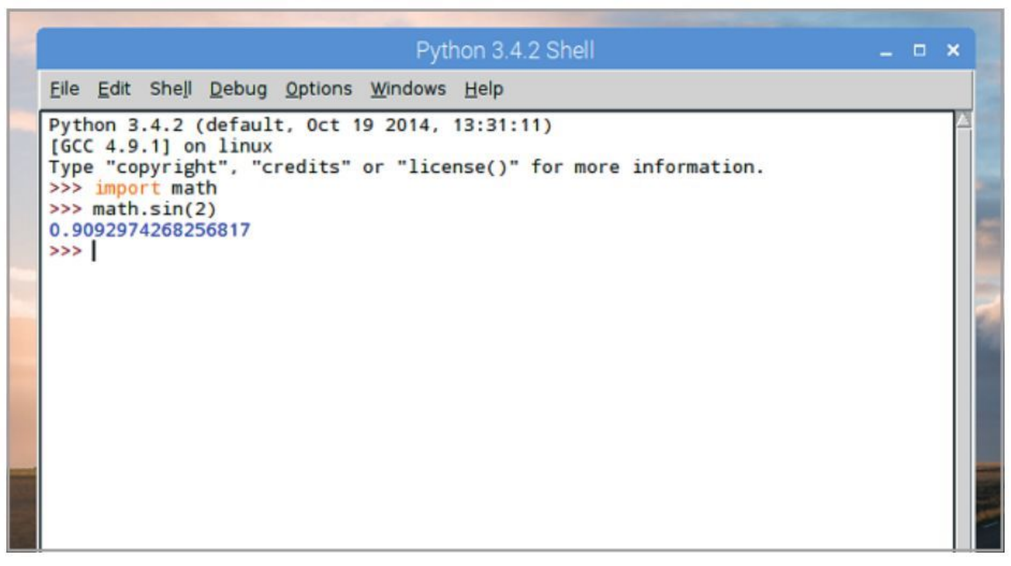
STEP 1 The Math module is built-in to Python 3; so there's no need to PIP install it. As with the other modules present, you can import the module's function by simply entering `import math` into the Shell, or as part of your code in the Editor.



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:

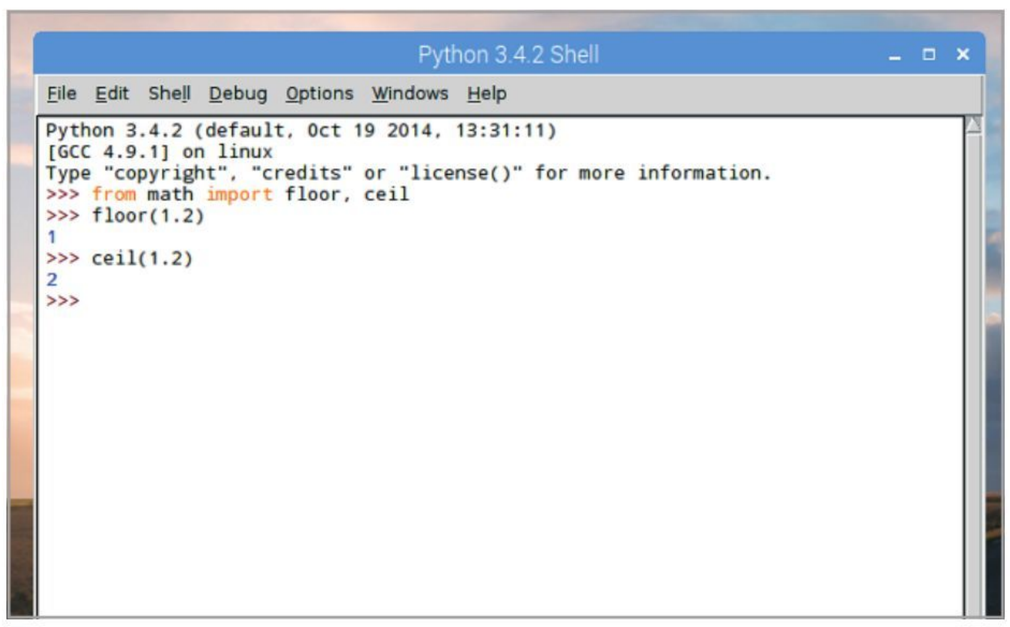
```
math.sin(2)
```

This displays the sine of 2.



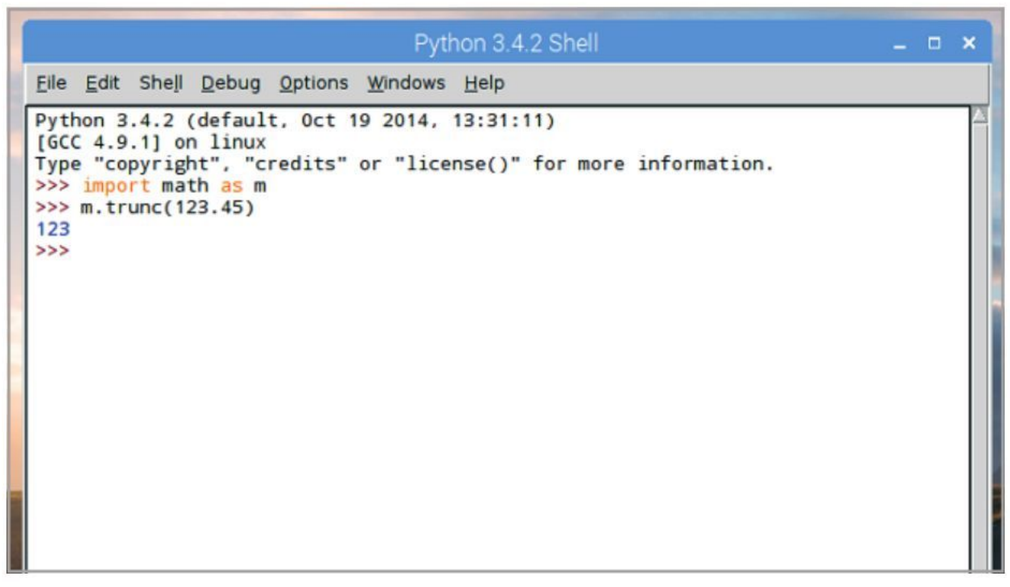
STEP 3 As you will no doubt be aware by now, if you know the name of the individual functions within the module you can specifically import them. For instance, the Floor and Ceil functions round down and up a float:

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



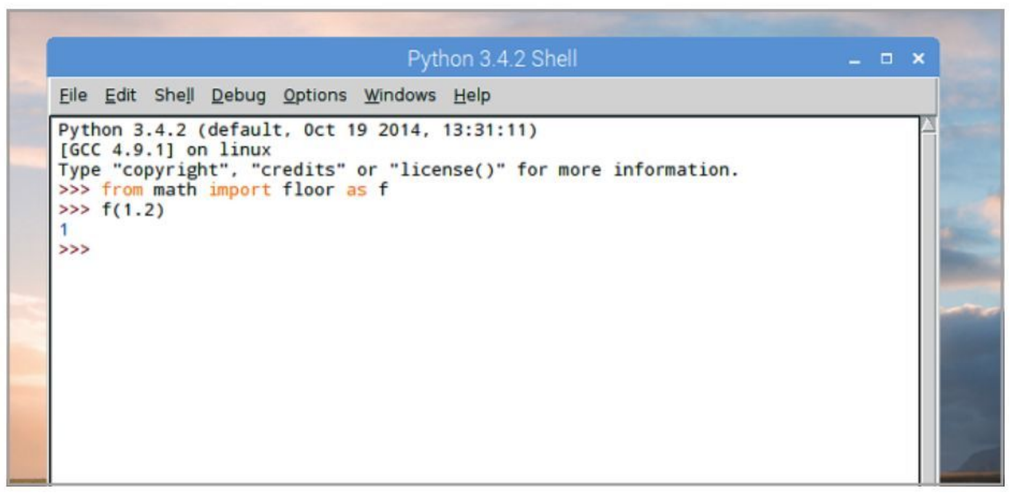
STEP 4 The Math module can also be renamed as you import it, as with the other modules on offer within Python. This often saves time, but don't forget to make a comment to show someone else looking at your code what you've done:

```
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:

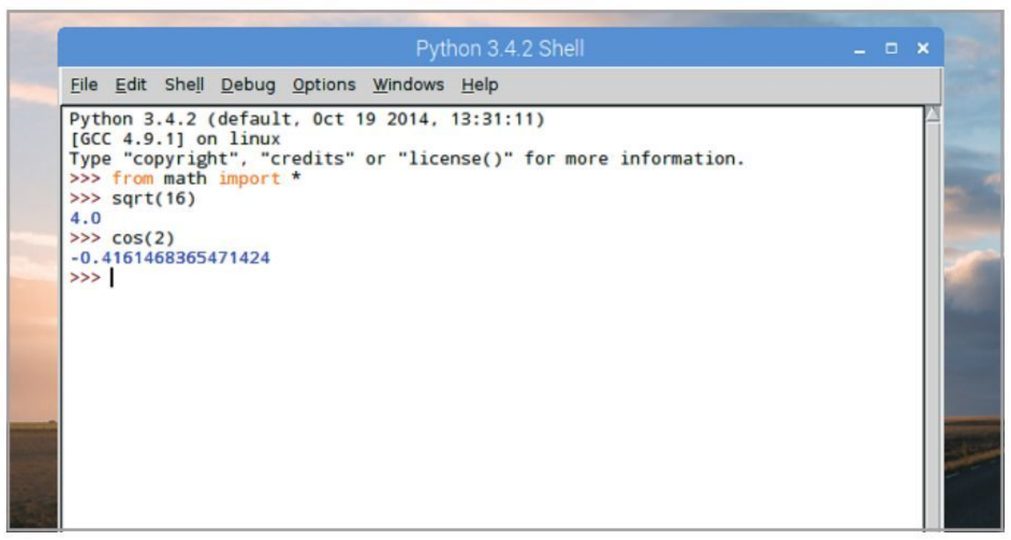
```
from math import floor as f
f(1.2)
```



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.



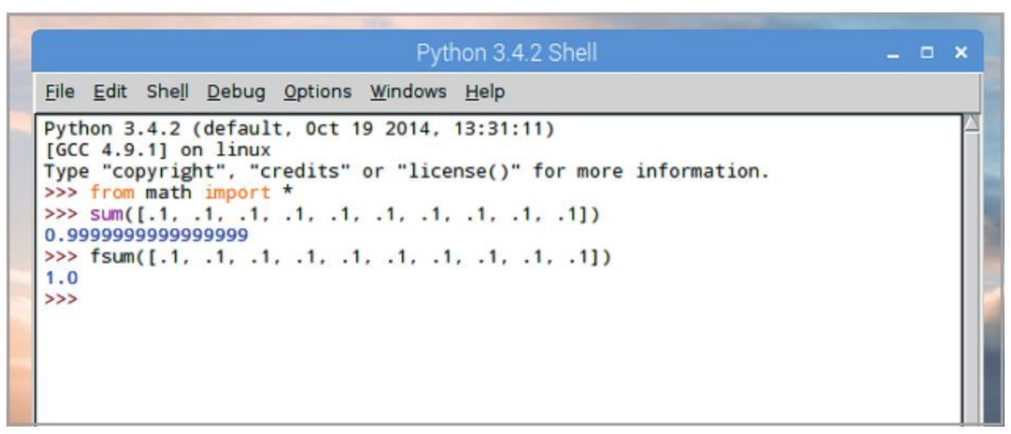
STEP 7 Interestingly, some functions within the Math module are more accurate, or to be more precise are designed to return a more accurate value, than others. For example:

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

will return the value of 0.999999999. Whereas:

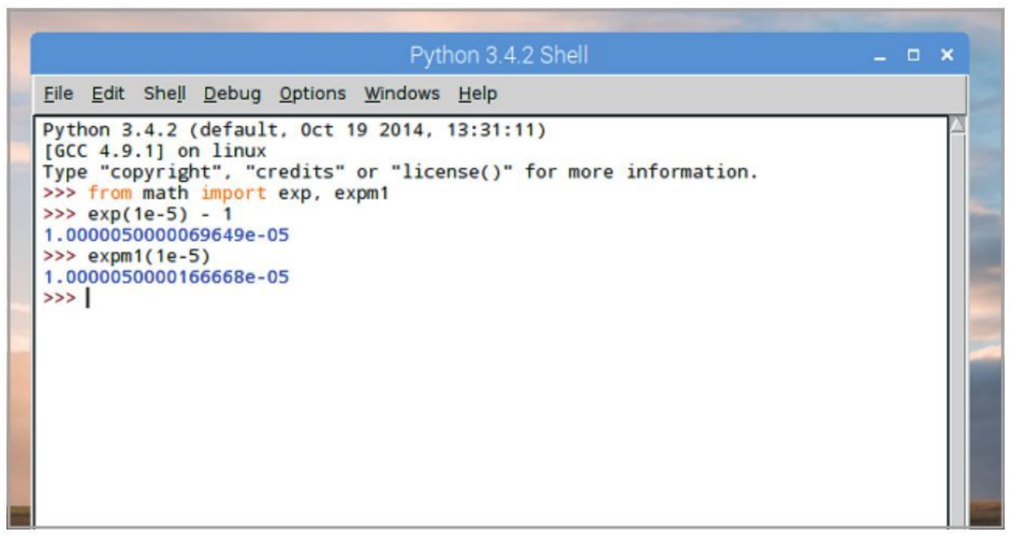
```
fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
```

returns the value of 1.0.



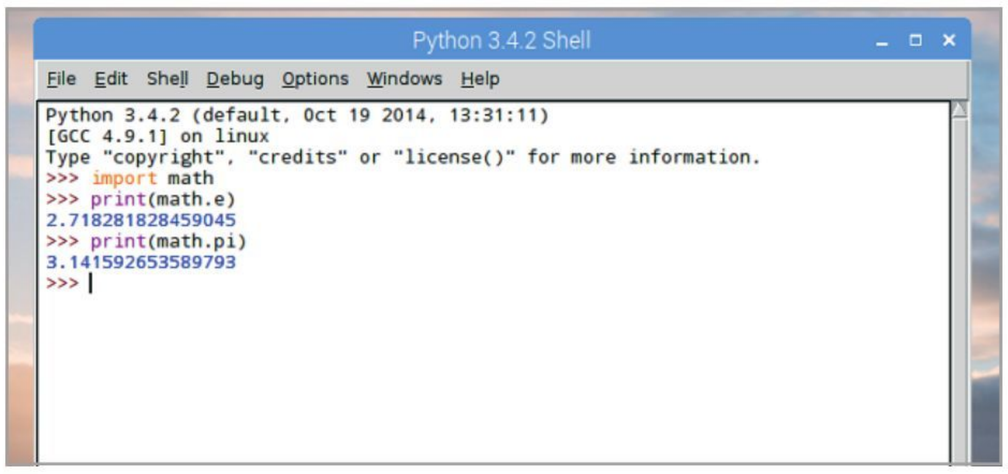
STEP 8 For further accuracy, when it comes to numbers the exp and expm1 functions can be used to compute precise values:

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

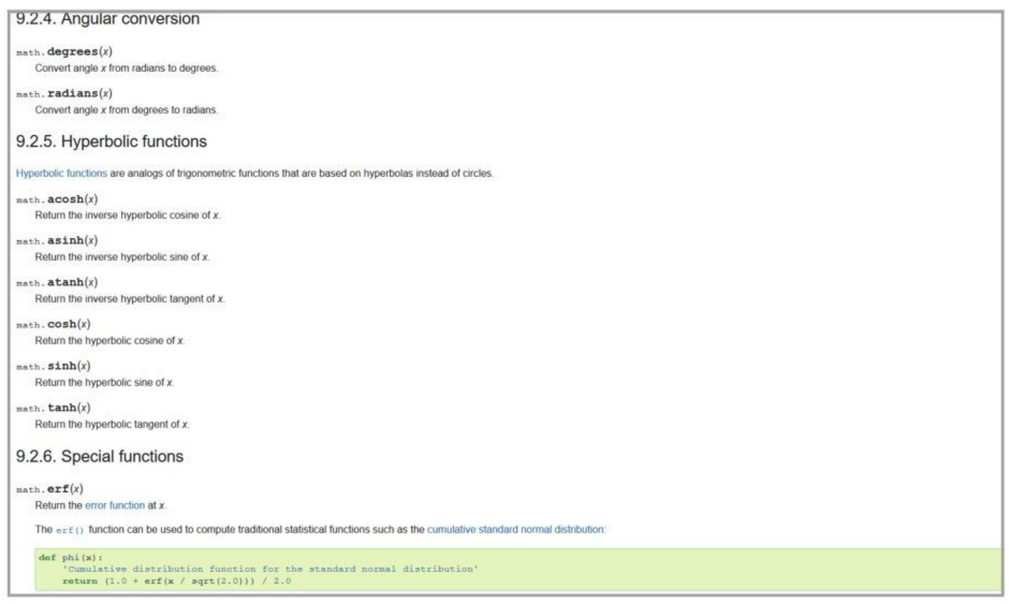


STEP 9 This level of accuracy is really quite impressive, but quite niche for the most part. Probably the two most used functions are **e** and **Pi**, where e is the numerical constant equal to 2.71828 (where the circumference of a circle is divided by its diameter):

```
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 of trigonometric functions that are based on hyperbolas instead of 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

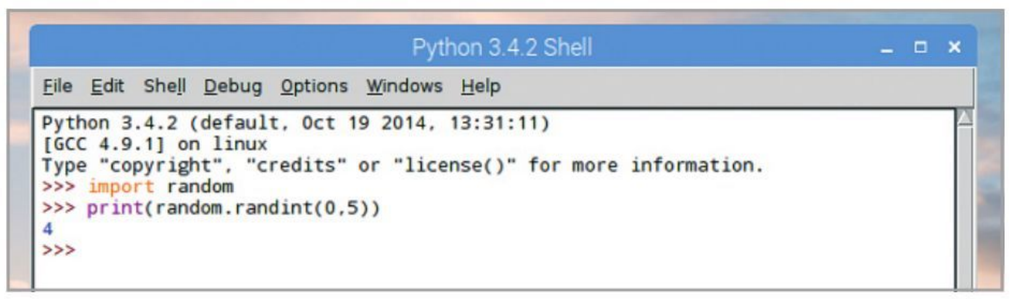
The Random module is one you will likely come across many times in your Python programming lifetime; as the name suggests, it's designed to create random numbers or letters. However, it's not exactly random but it will suffice for most needs.

RANDOM NUMBERS

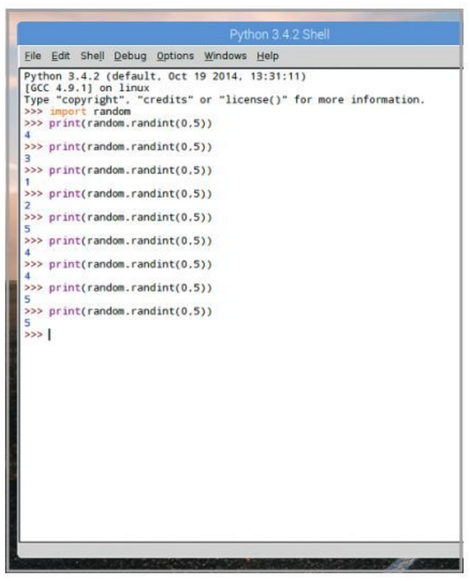
There are numerous functions within the Random module, which when applied can create some interesting and very useful Python programs.

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))
```



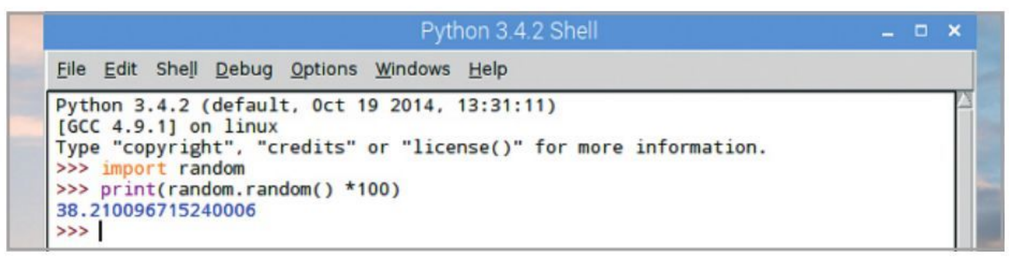
STEP 2 In our example the number four was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.



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)
```

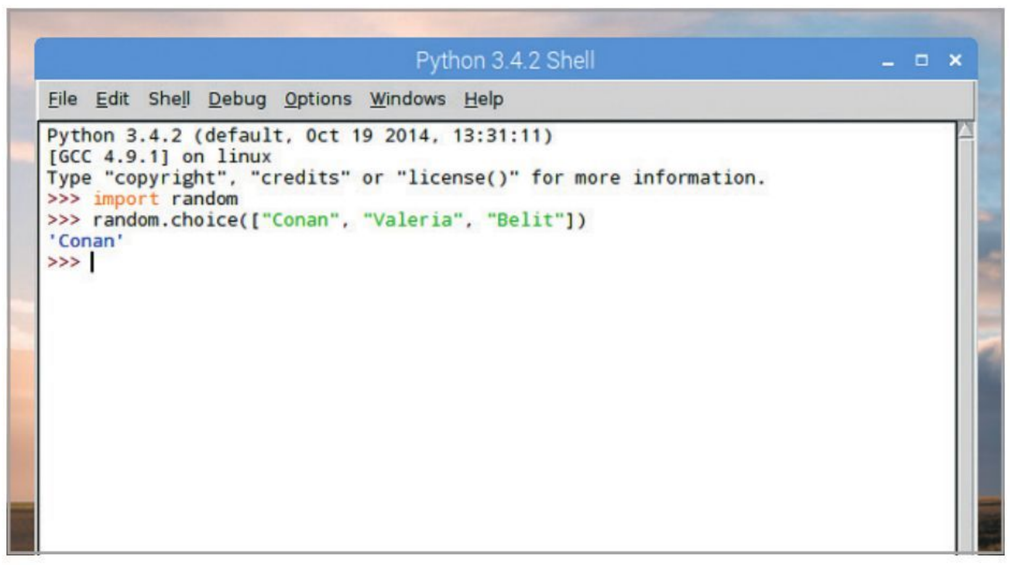
Will display a floating point number between 0 and 100, to the tune of around fifteen decimal points.



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:

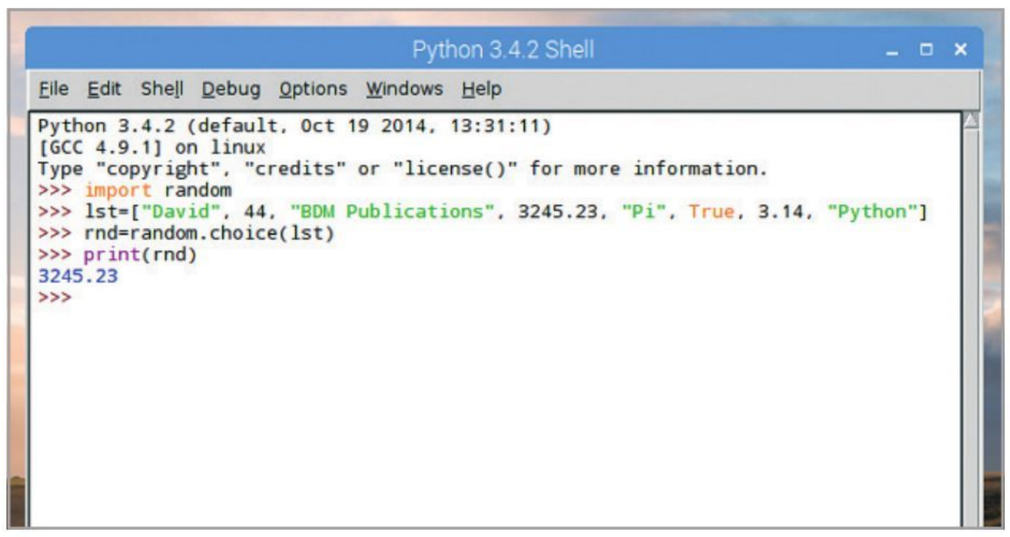
```
import random
random.choice(["Conan", "Valeria", "Belit"])
```

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)
```

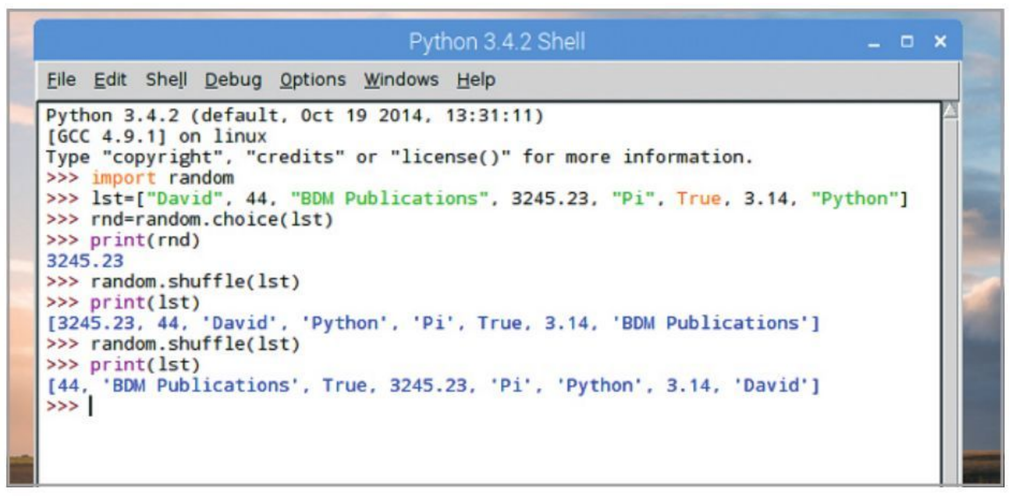


STEP 6

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.

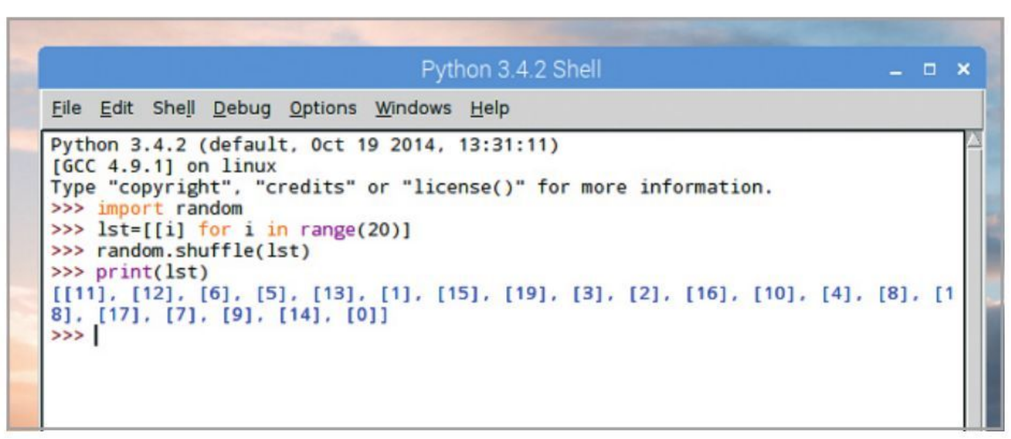


STEP 7

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.

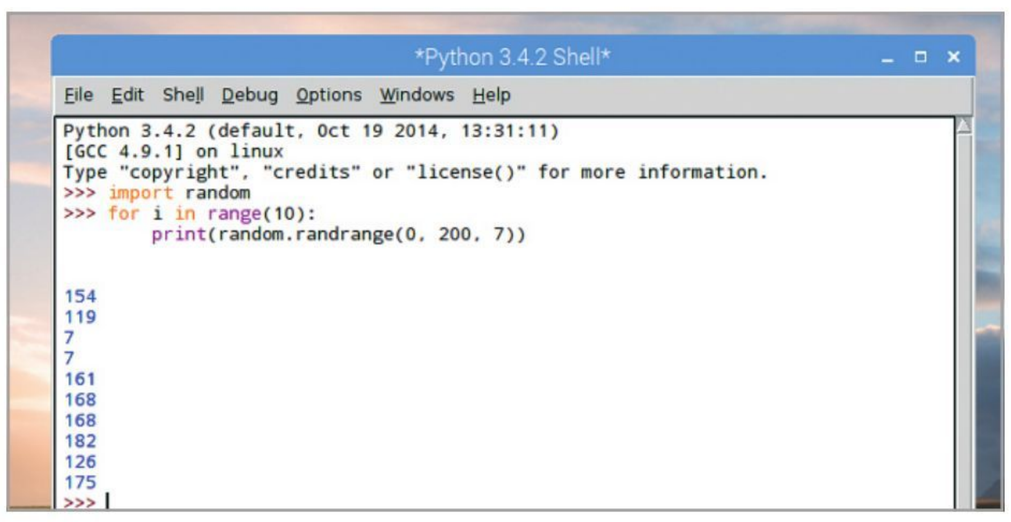


STEP 8

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