Using Modules

Modules are where you can take Python programming to new heights. You can create your own modules, or you can 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.

84 Calendar Module
86 OS Module
88 Random Module
90 Tkinter Module
92 Pygame Module
96 Create Your Own Modules

Calendar Module

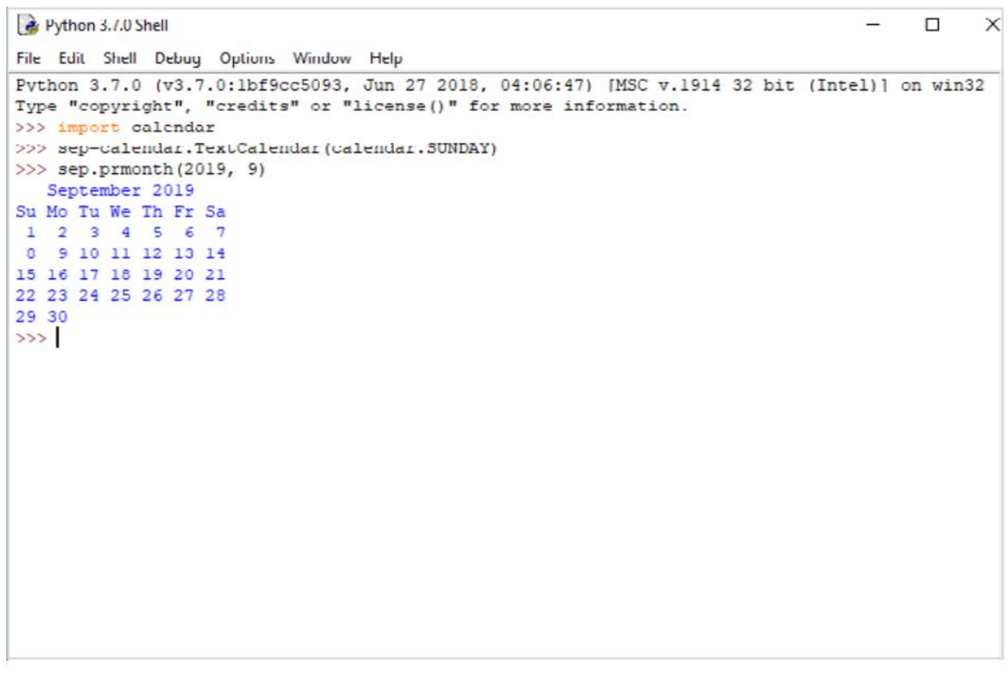
Beyond the Time module, the Calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the time module-like format, you can actually call up a wall-calendar type display.

WORKING WITH DATES

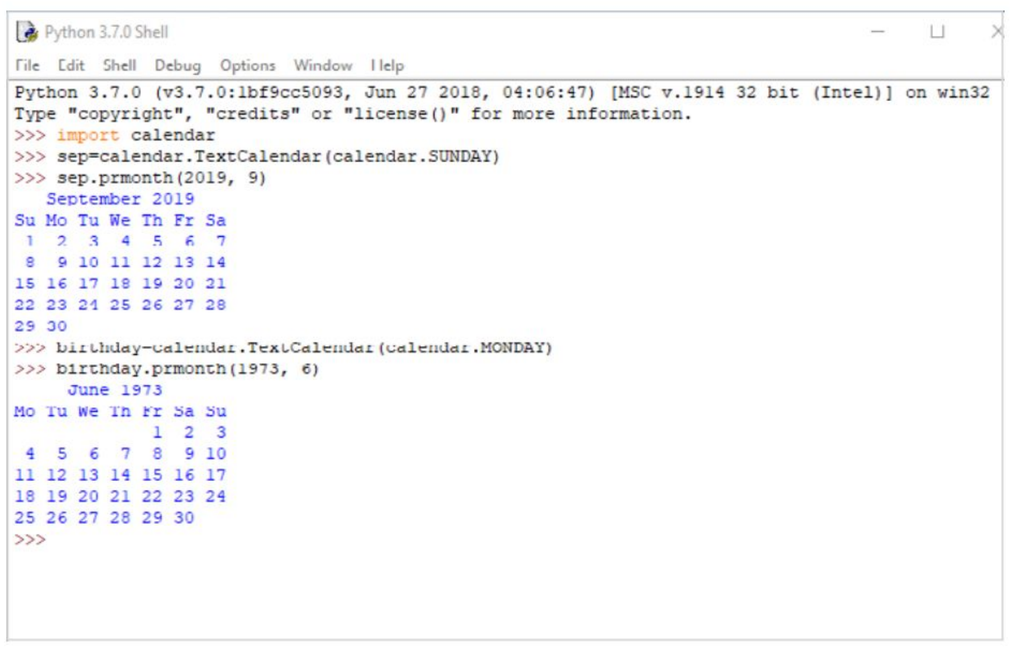
The Calendar module is built into Python 3. However, if, it's not installed, you can add it using **pip install calendar**, as a Windows administrator, or sudo **pip install calendar** for Linux and macOS.

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.



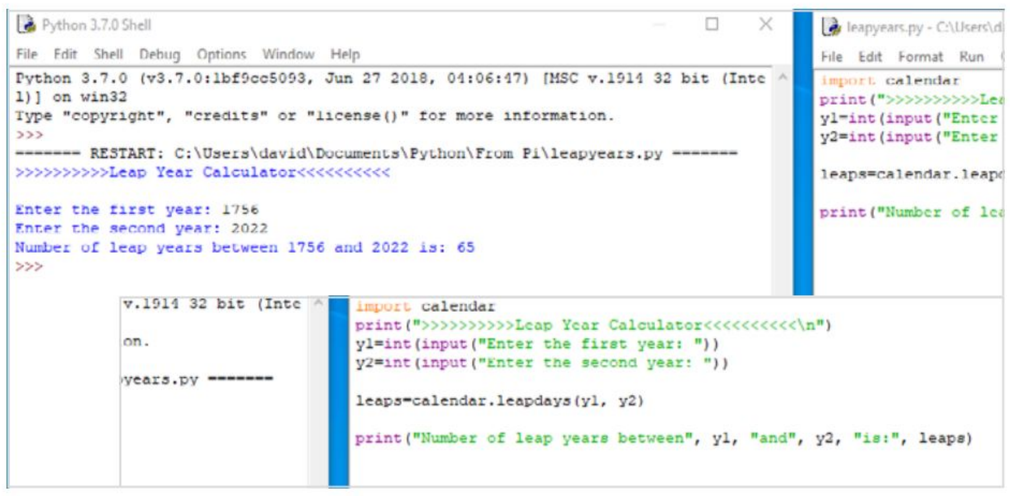
STEP 3 There are numerous functions, within the Calendar module, that may be of interest to you when forming your own code. For example, you can display the number of leap years between two specific years:

```
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)
```

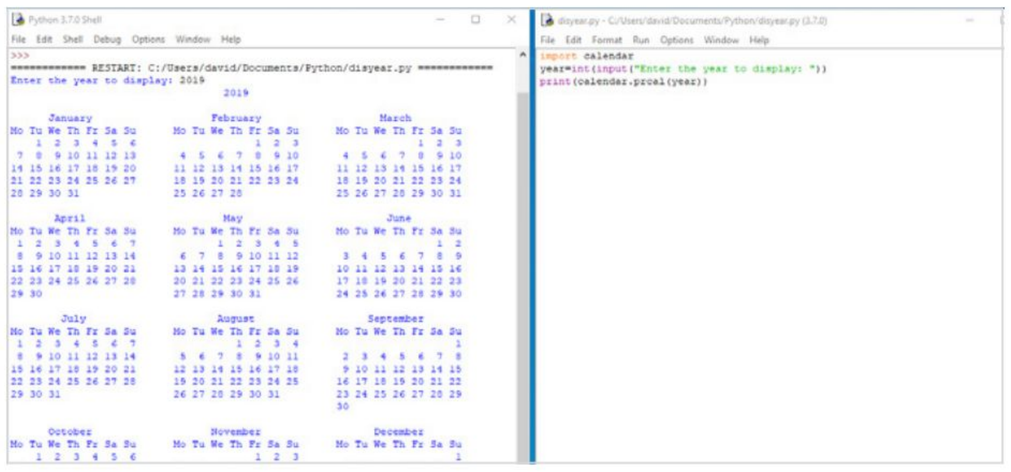


STEP 5

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))
```

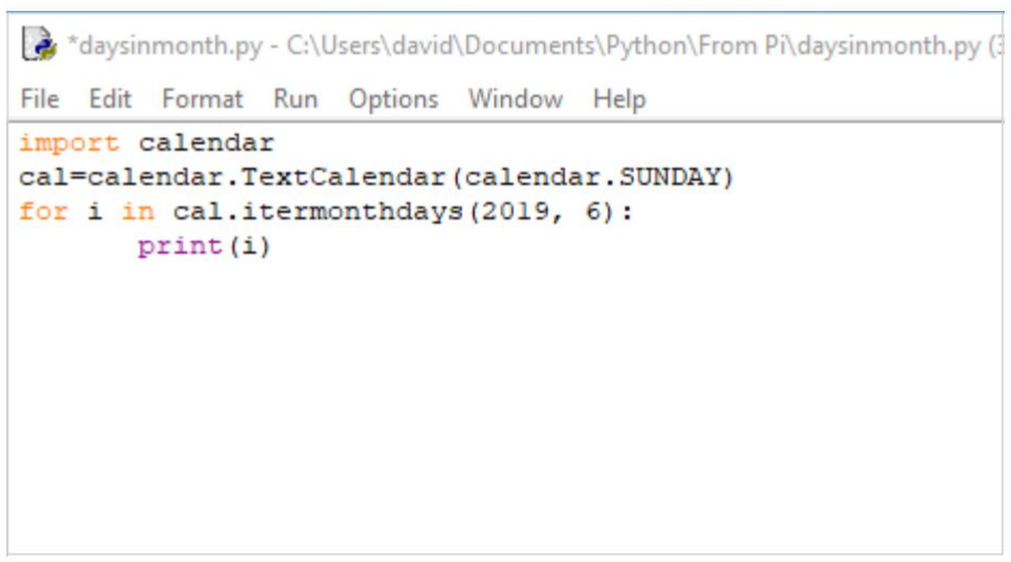
We're sure you'll agree that's quite a handy bit of code to have to hand.



STEP 6

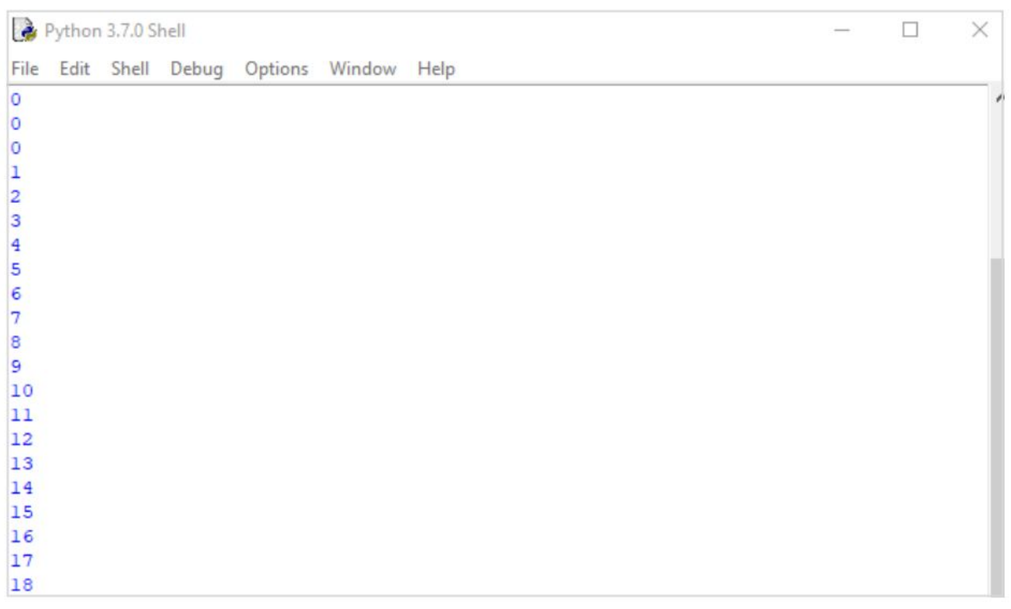
Interestingly we can also list the number of days in a month by using a simple for loop:

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



STEP 7

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

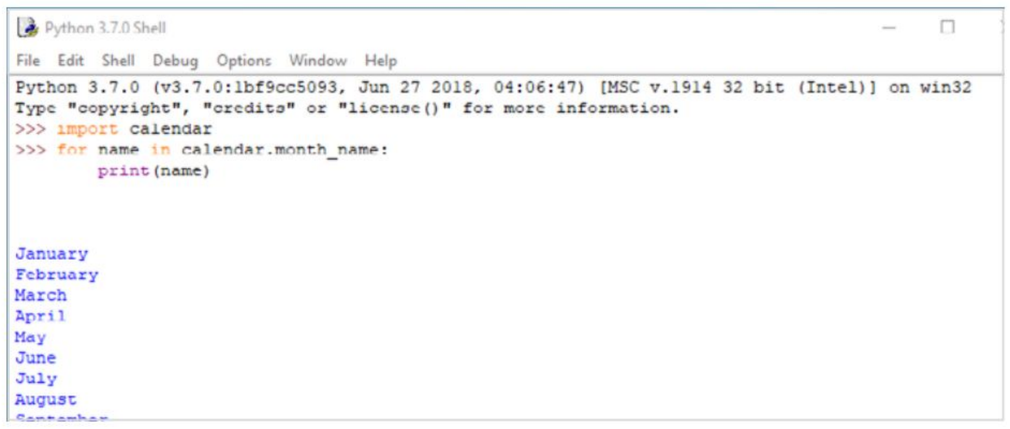


STEP 8

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)
```

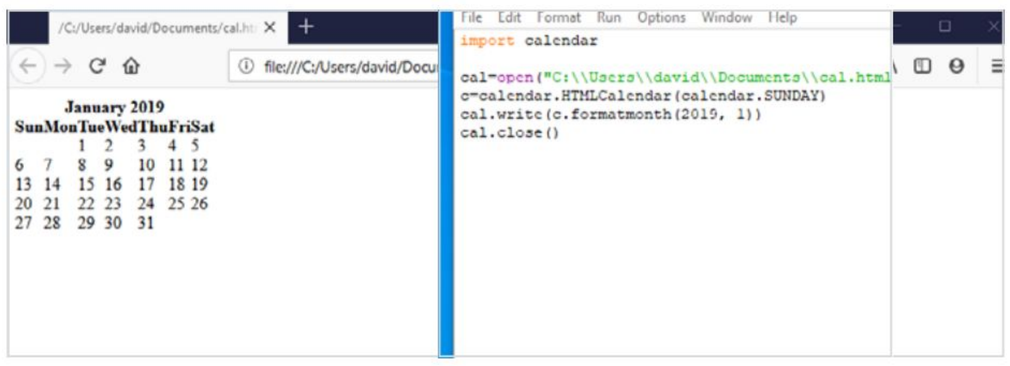


STEP 9

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("C:\\Users\\david\\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.

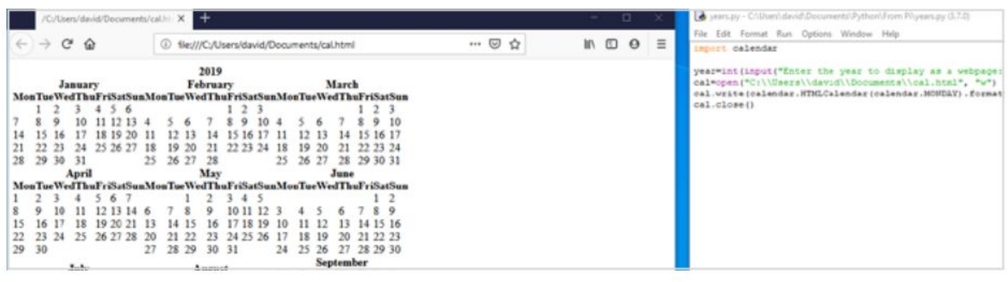


STEP 10

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("C:\\Users\\david\\Documents\\cal.html", "w")
cal.write(calendar.HTMLCalendar(calendar.MONDAY).
formatyear(year))
cal.close()
```

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



OS Module

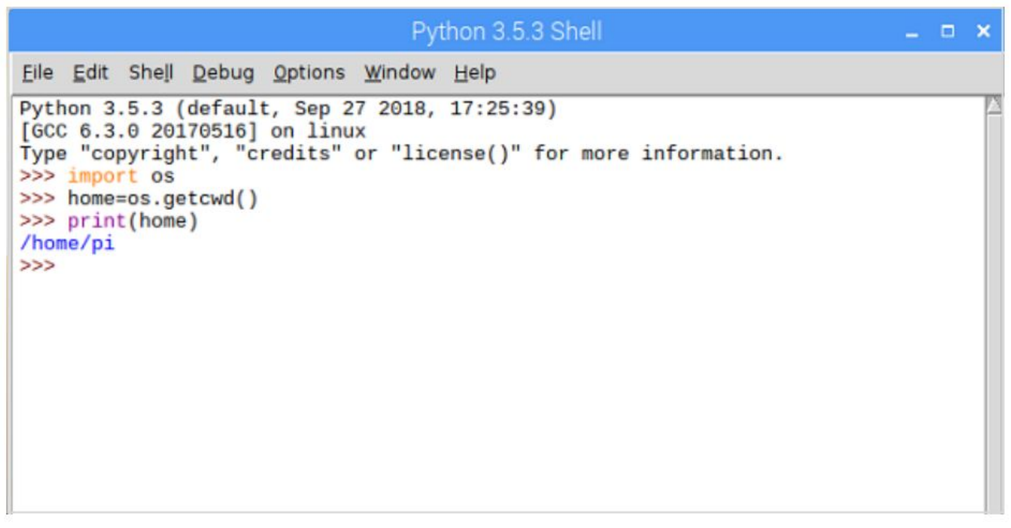
The OS module allows you to interact directly with the built-in commands found in your operating system. The commands can vary, depending on the OS on which you're running the module, as some will work with Windows whereas others will work with Linux and macOS.

INTO THE SYSTEM

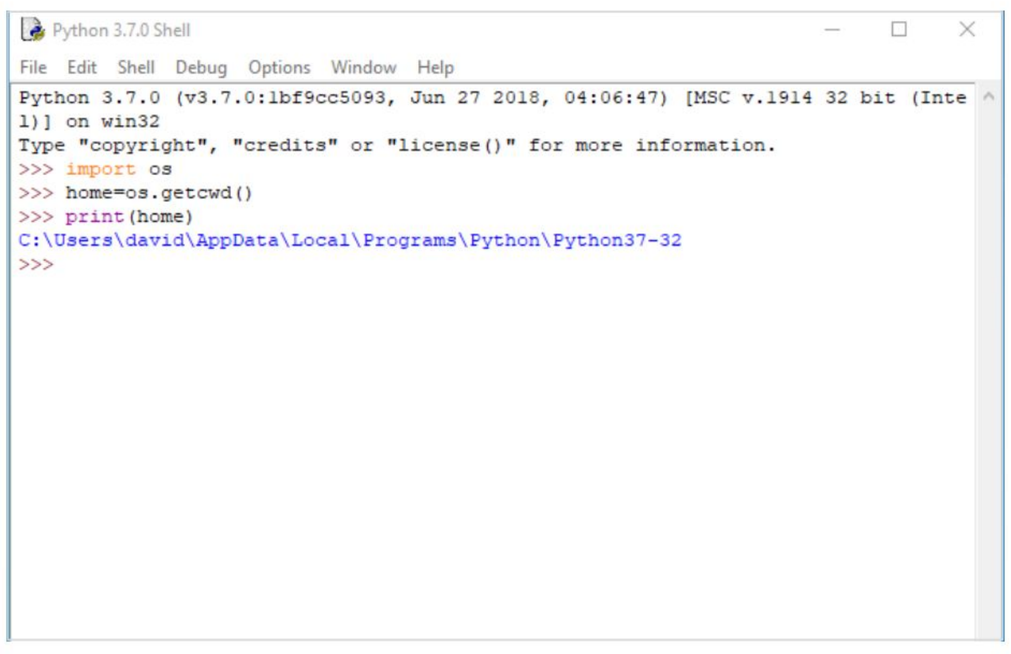
One of the primary features of the OS module is the ability to list, move, create, delete and otherwise interact with files stored on the system; making it the perfect module for backup code.

STEP 1 We 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:

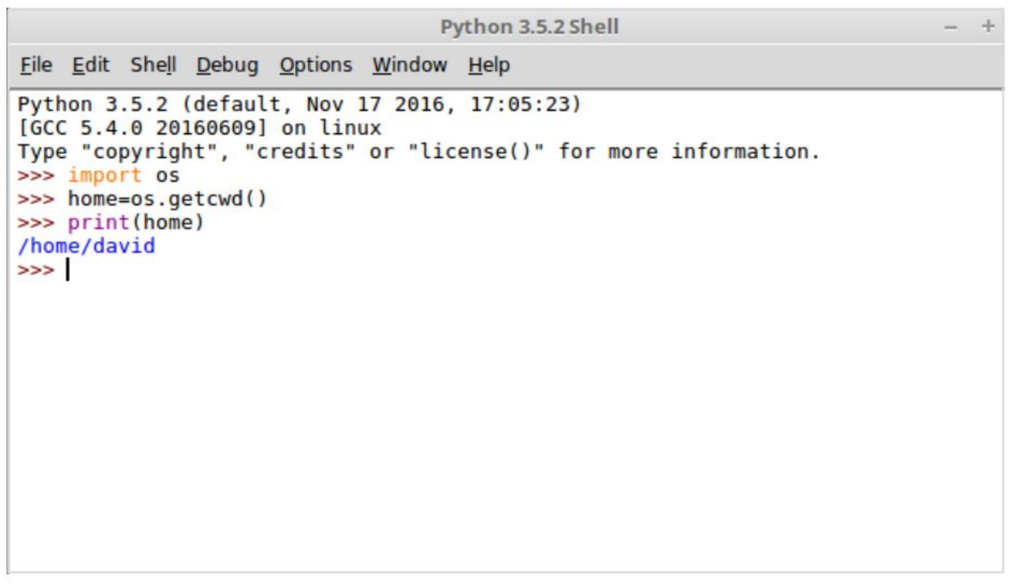
```
import os
home=os.getcwd()
print(home)
```



STEP 2 The returned result, from printing the variable home, is the current user's home folder on the system. In Step 1 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 would output: **C:\Users\david\AppData\Local\Programs\Python\Python37-32**.

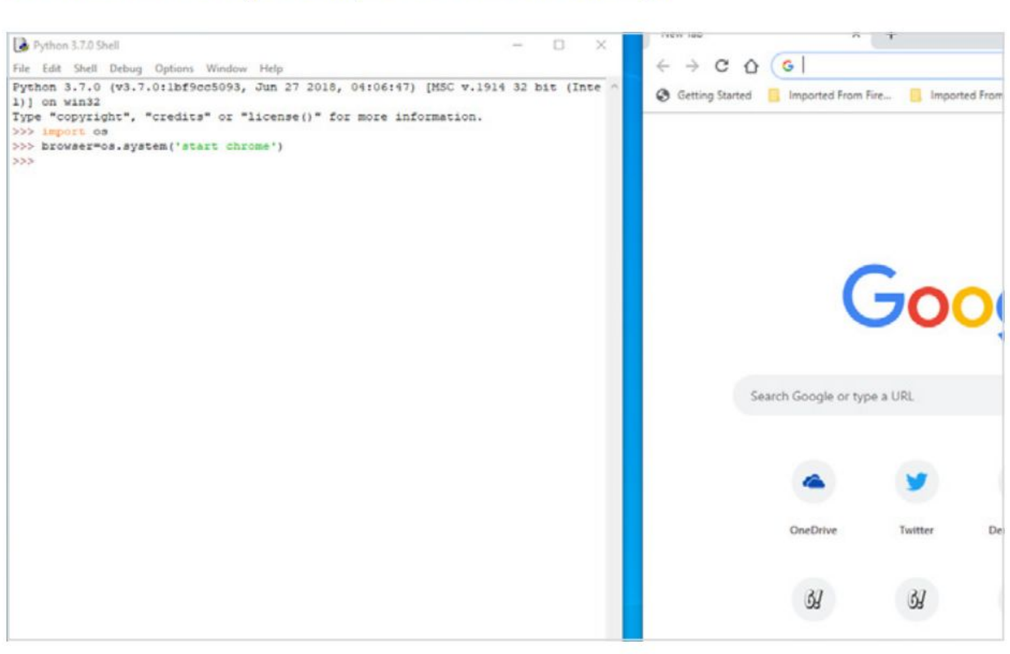


STEP 3 The Windows output is different as that's the current working directory of Python, as determined by the system. As you suspect, the **os.getcwd()** function is asking Python to retrieve the Current Working Directory. Linux users will see something along the same lines as the Raspberry Pi, as will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



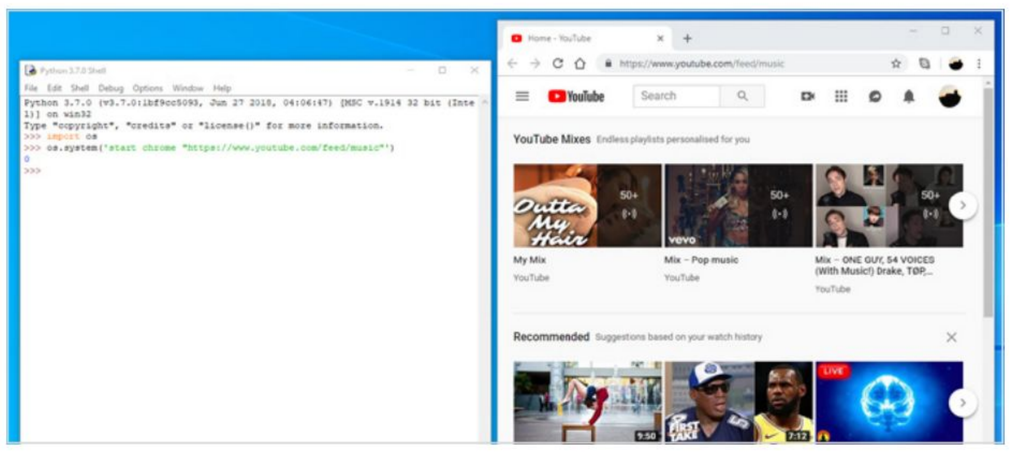
STEP 4 Another interesting element to the OS module is its ability to launch programs that are installed in the host system. For instance, if we wanted to launch the Chromium Web Browser from within a Python program we can use the command:

```
import os
browser=os.system('start chrome')
```



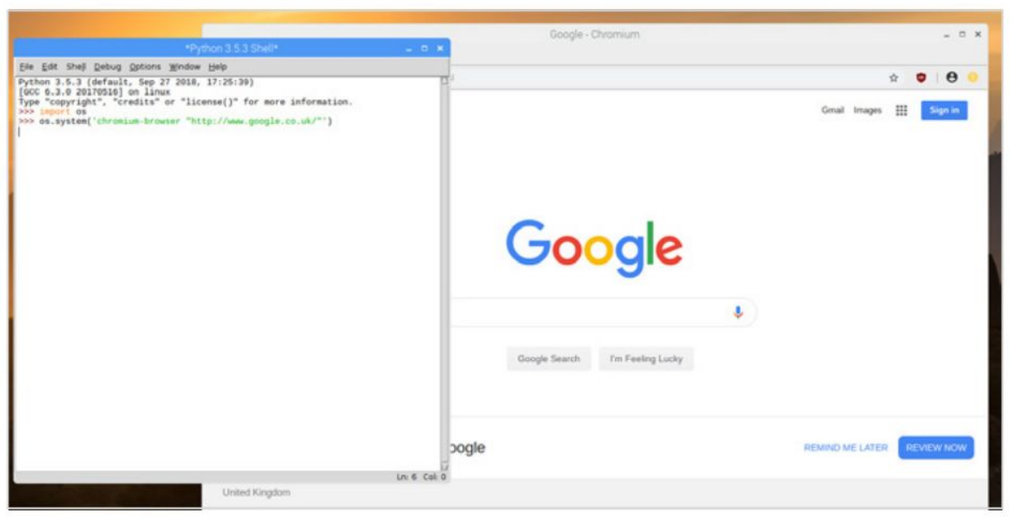
STEP 5 The os.system() function is what allows interaction with external programs – you can even call up previous Python programs using this method. You will obviously need to know the full path and program file name for it to work successfully. However, you can also use the following:

```
import os
os.system(‘‘start chrome “https://www.youtube.com/feed/music”’)
```



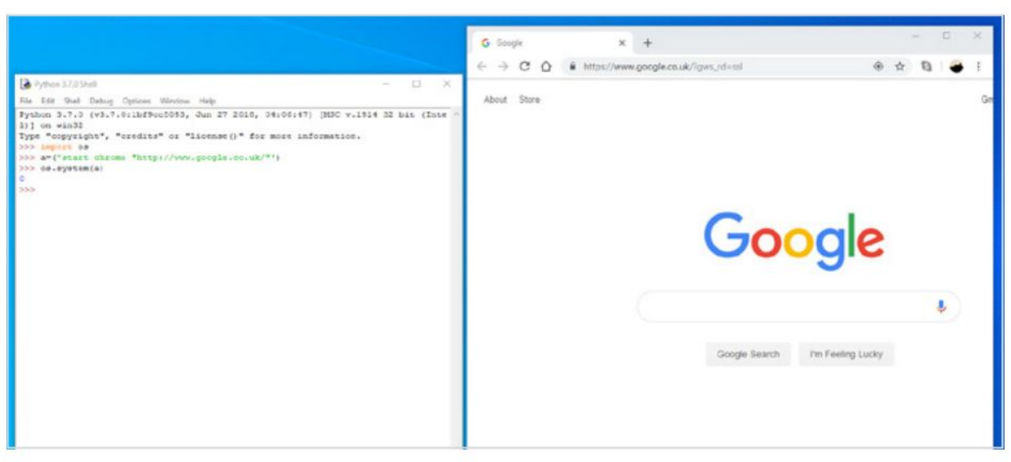
STEP 6 To show that the OS module works roughly the same across all platforms, we need to specify the name of the app according to the OS on which you’re working. For example, on a Raspberry Pi you would enter:

```
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 open webpages:

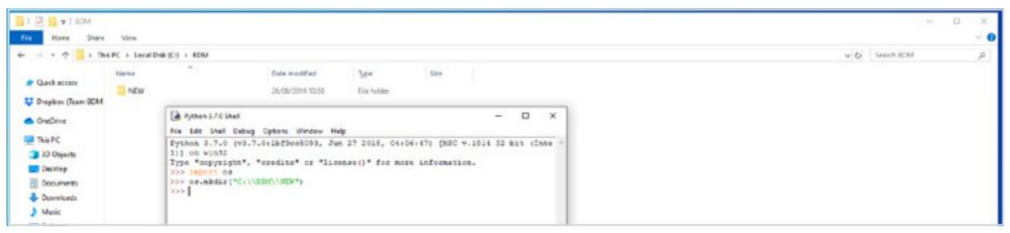
```
import os
a=(‘start chrome “http://www.google.co.uk”’)
os.system(a)
```



STEP 8 Being able 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(“C:\\BDM\\NEW”)
```

This creates a new directory within the specified Directory (C:\BDM\), named according to the object in the mkdir function (C:\BDM\NEW).

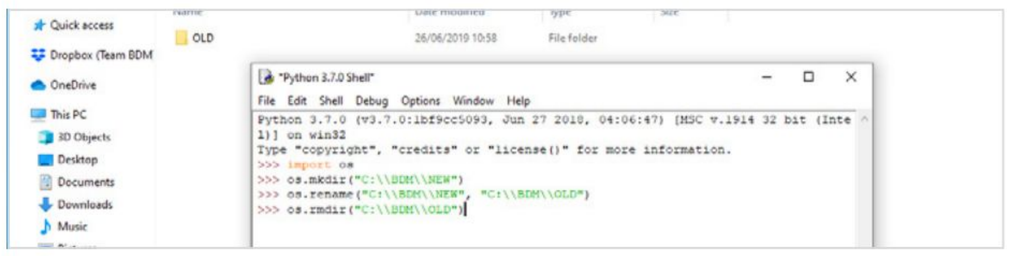


STEP 9 You can also rename any directories you’ve created by entering:

```
import os
os.rename(“C:\\BDM\\NEW”, “C:\\BDM\\OLD”)
```

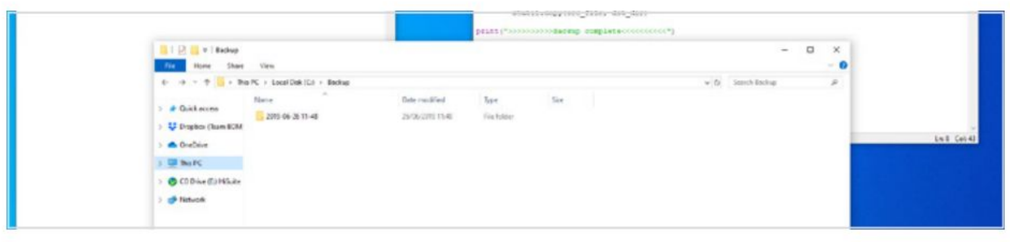
Or delete them…

```
import os
os.rmdir(“C:\\BDM\\OLD”)
```



STEP 10 Another module that goes together with OS is shutil. We 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

t=str(time.strftime(“%Y-%m-%d %H-%M”))
root_src_dir = r’C:\\BDM\\Documents\\’    #Path to the source directory
root_dst_dir = ‘C:\\Backup\\’ + t #Path to the backup directory

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<<<<<<<<<<”)
```



Random Module

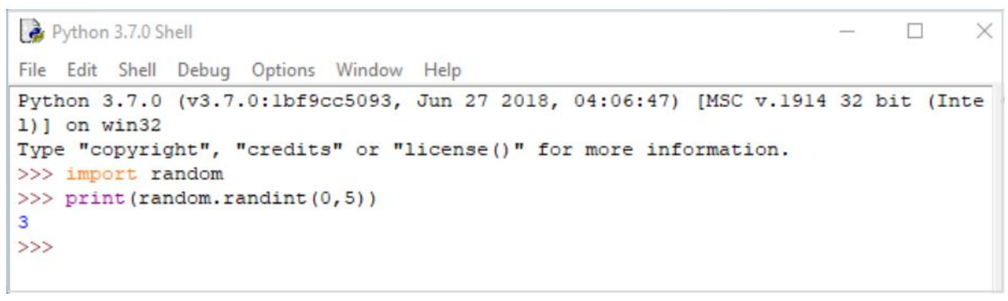
The Random module is one you will likely come across many times in your Python programming lifetime. As the name suggests, it's designed to create a random set of numbers from a given set. Although it's not exactly random, there is a pattern behind it, it will suffice for most needs.

RANDOM NUMBERS

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

STEP 1 As with other modules, you'll 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))
```

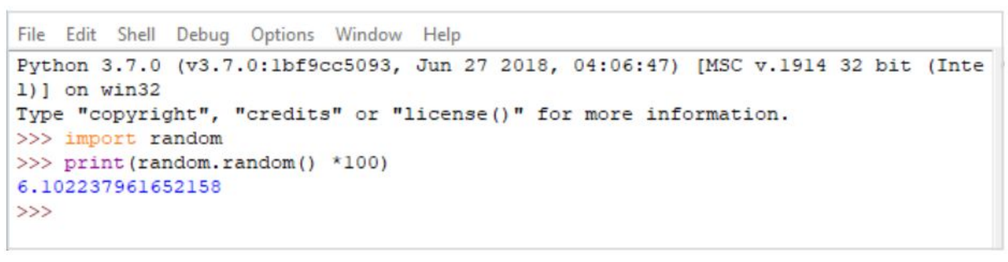


STEP 2 In our example the number three was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.

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

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

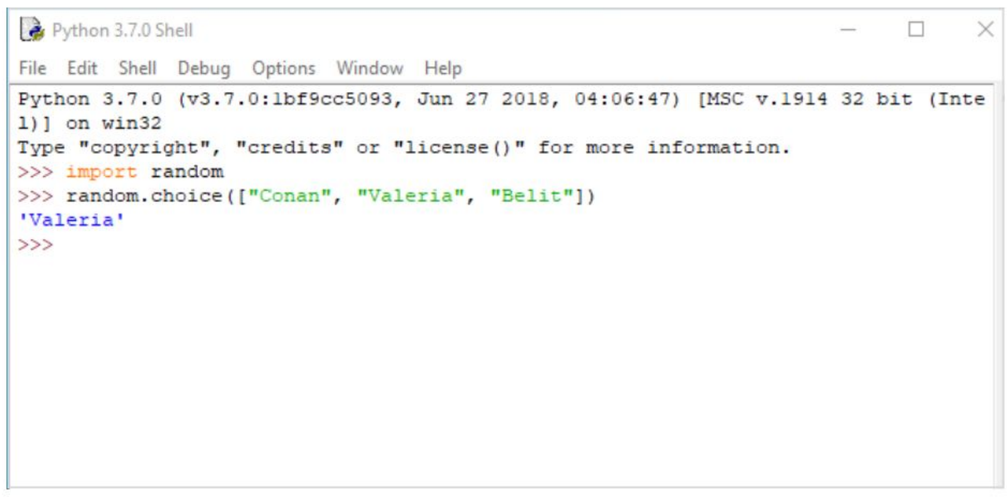
Will display a floating point number, between 0 and 100, to the tune of around fifteen decimal points.



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

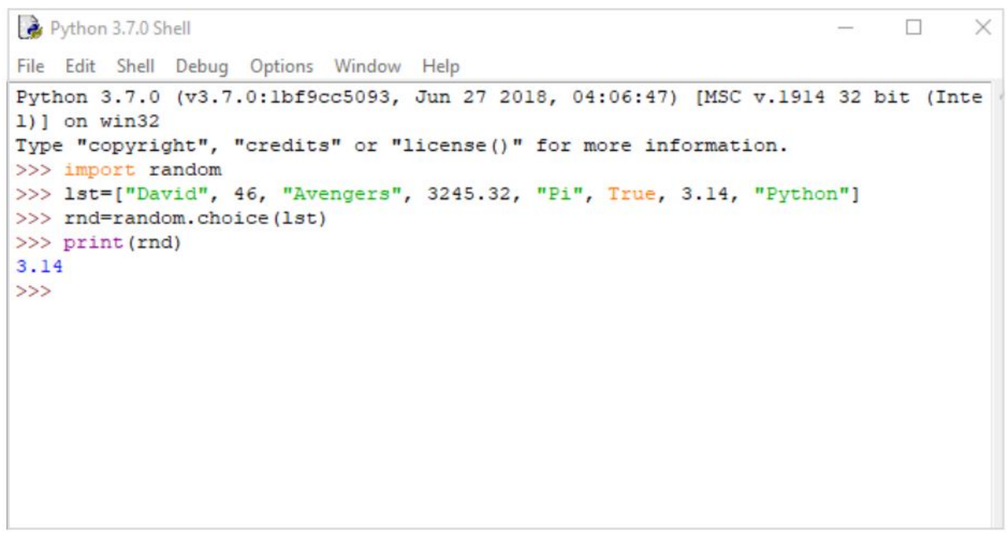
```
import random
random.choice(["Conan", "Valeria", "Belit"])
```

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



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

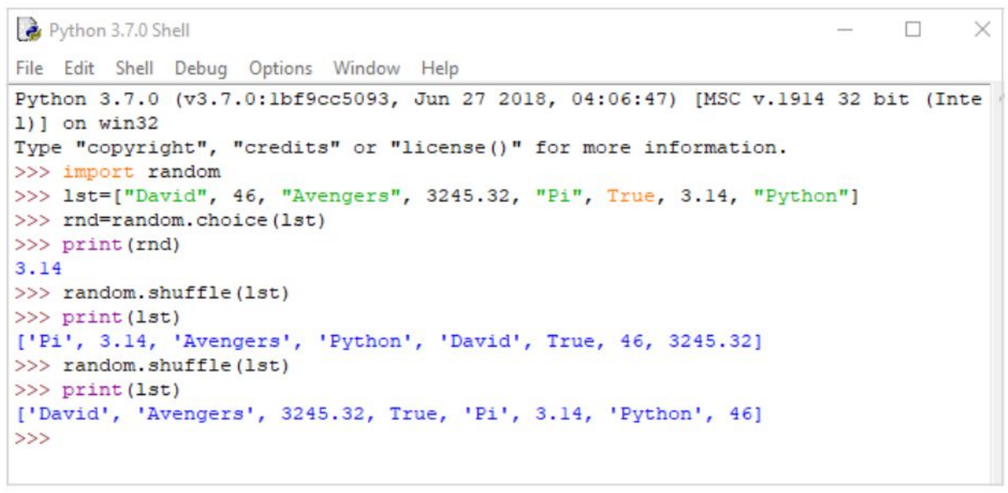
```
import random
lst=["David", 46, "Avengers", 3245.32, "Pi", True,
3.14, "Python"]
rnd=random.choice(lst)
print(rnd)
```



STEP 6 Interestingly, we 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)
```

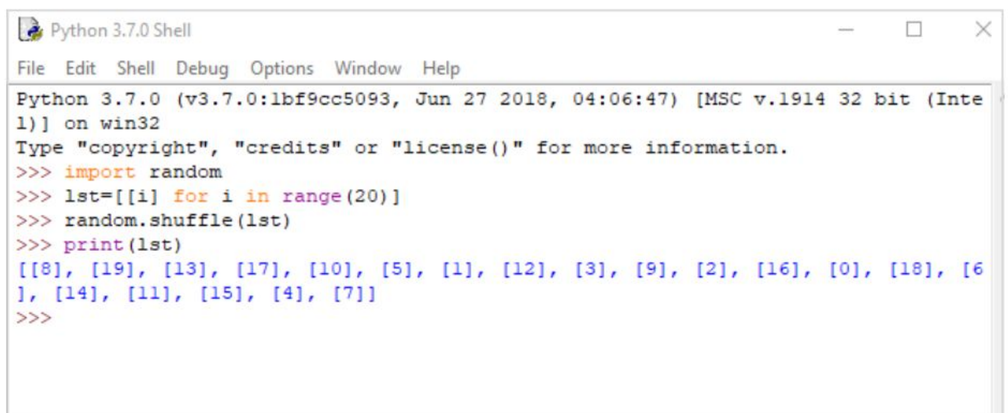
This way, you can keep shuffling the list before displaying a random item from it.



STEP 7 Using shuffle, we 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)
```

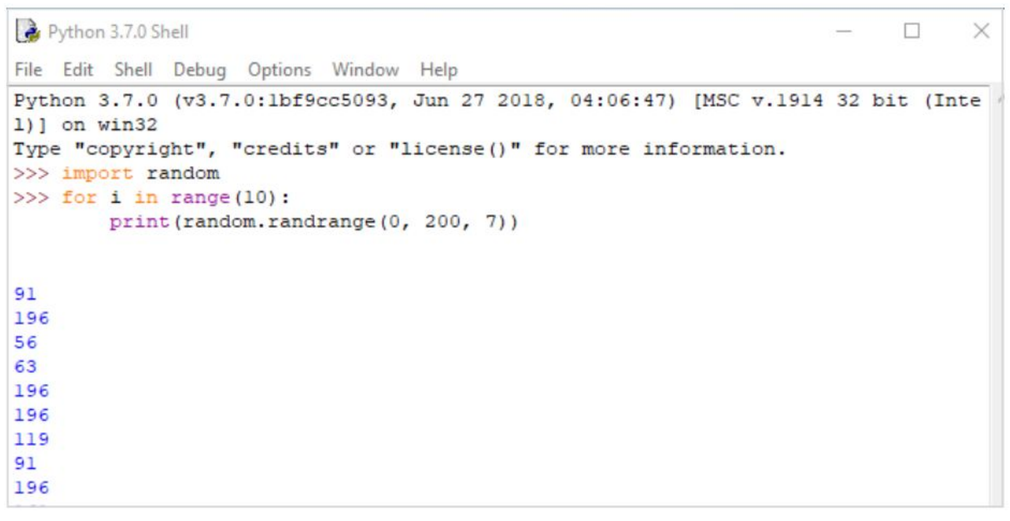
Keep shuffling the list, and you'll have a different selection of items from 0 to 20 every time.



STEP 8 We can also select a random number from a given range in steps, using the start, stop, step loop:

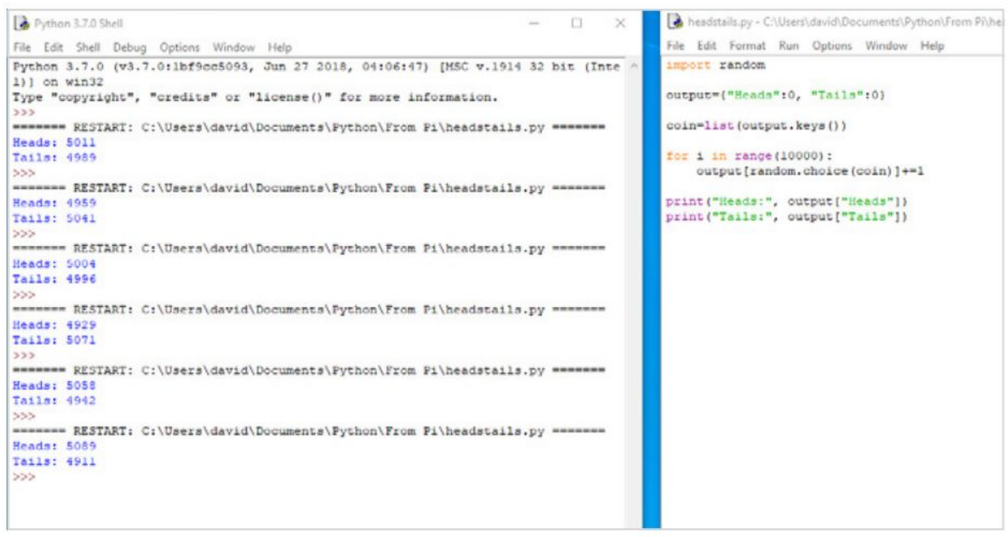
```
import random
for i in range(10):
      print(random.randrange(0, 200, 7))
```

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



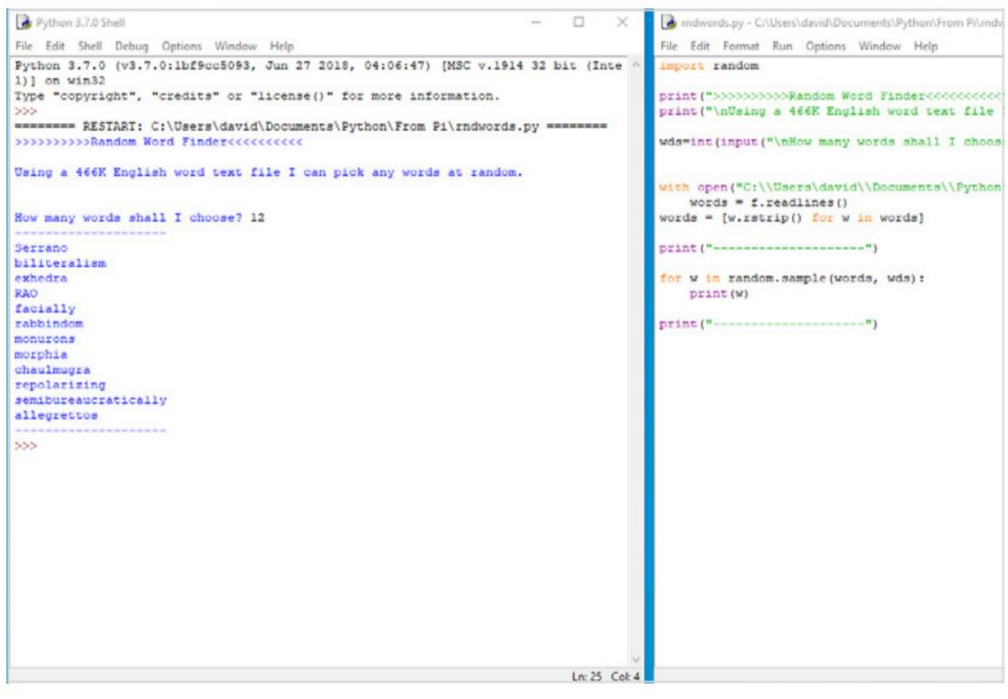
STEP 9 Let's use an example piece of code, which flips a virtual coin ten thousand times and counts how many times it'll land on Heads or Tails:

```
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, we can pluck a user generated number of words from the file (text file found at: https://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 then getting some code into it.

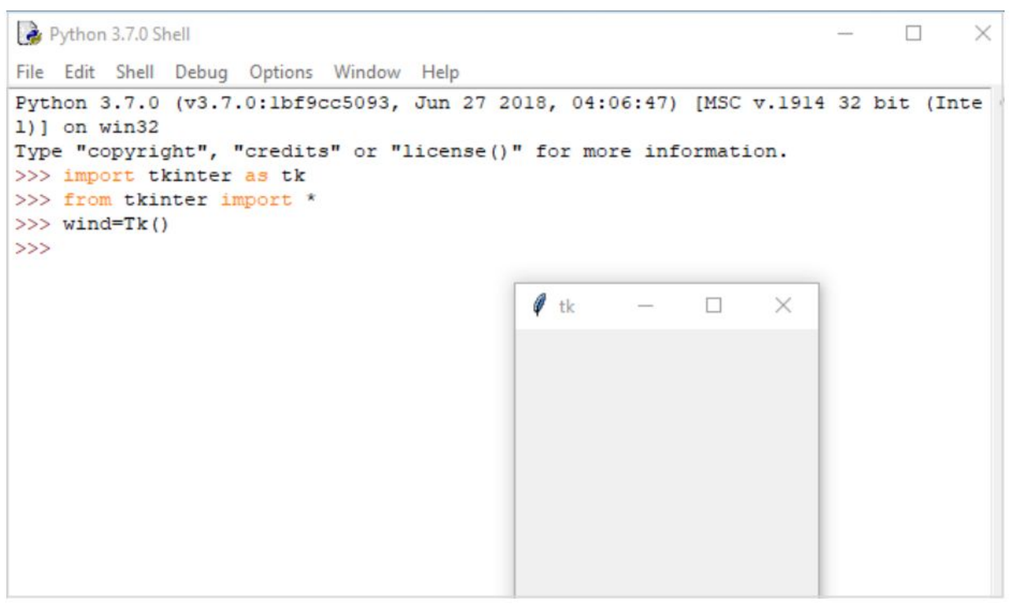
STEP 1 Tkinter is usually built into Python 3, however if it's available when you enter: `import tkinter`, then you'll need to **pip install tkinter** from the command prompt. We can start to import modules differently than before, to save on typing, by importing all their contents:

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

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

```
wind=Tk()
```

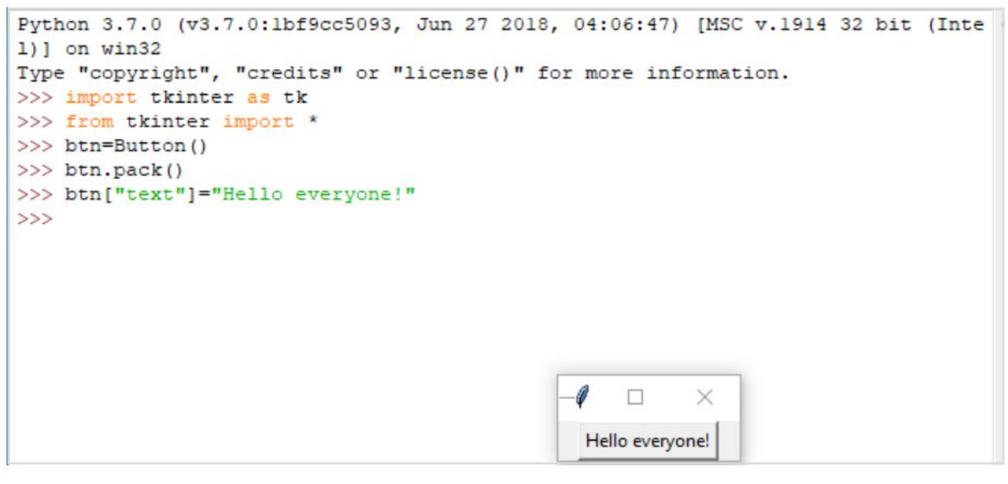
This creates a small, basic window. At this point, there's not much else to do but click the X in the corner to close the window.



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.

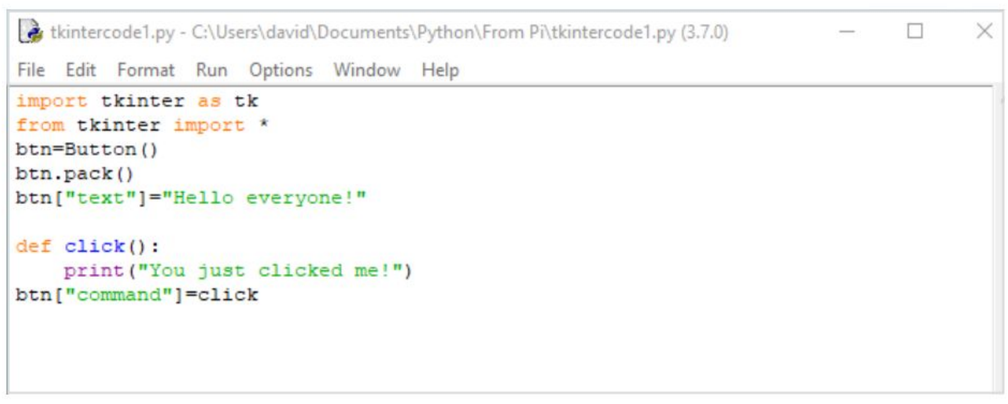


STEP 4 We can combine the above into a New File:

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

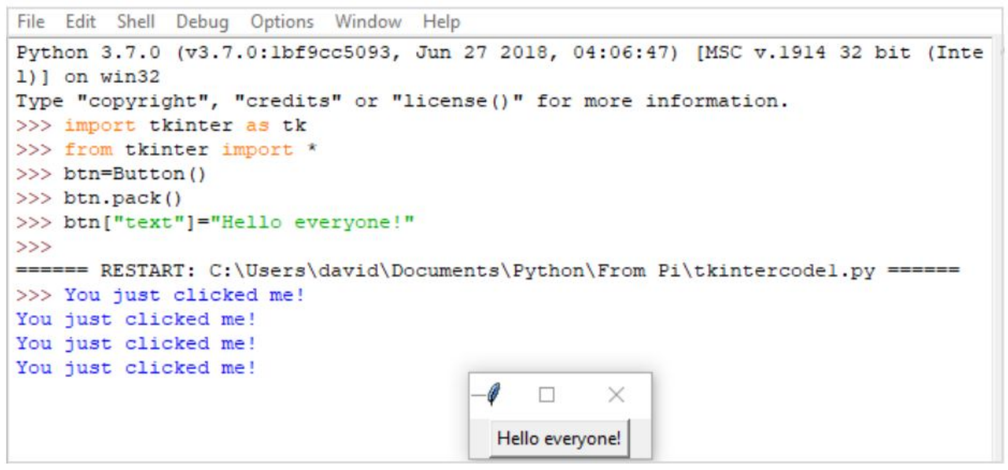
Then add some button interactions:

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



STEP 5

Save and execute the code from Step 5, and a window will appear 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.



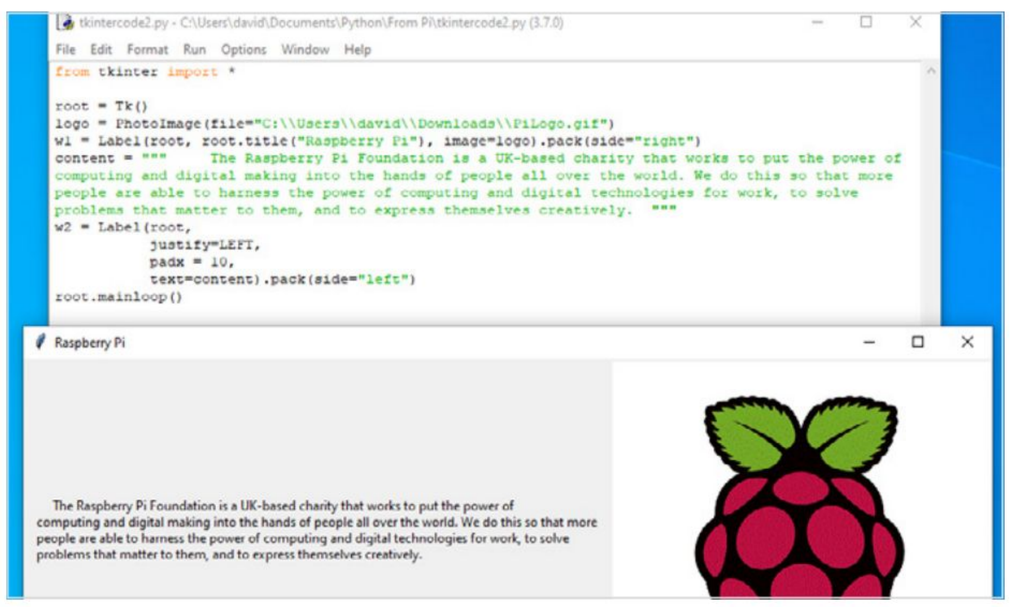
STEP 6

We 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 Raspberry Pi logo:

```
from tkinter import *

root = Tk()
logo = PhotoImage(file="C:\\Users\\david\\Downloads\\
PiLogo.gif")
w1 = Label(root, root.title("Raspberry Pi"),
image=logo).pack(side="right")
content = """     The Raspberry Pi Foundation is a
UK-based charity that works to put the power of
computing and digital making into the hands of
people all over the world. We do this so that more
people are able to harness the power of computing
and digital technologies for work, to solve
problems that matter to them, and to express
themselves creatively.  """
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 Raspberry Pi'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

We 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

And we can 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.

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

def var_states():
   print("Warrior: %d,\nMage: %d" % (var1.get(), var2.get(), var3.get(), var4.get(), var5.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)
var3 = IntVar()
Checkbutton(root, text="Ranger", variable=var3).grid(row=3, sticky=W)
var4 = IntVar()
Checkbutton(root, text="Barbarian", variable=var4).grid(row=4, sticky=W)
var5 = IntVar()
Checkbutton(root, text="Paladin", variable=var5).grid(row=5, sticky=W)

Button(root, text='Quit', command=root.destroy).grid(row=7, sticky=W, pady=4)
Button(root, text='Show', command=var_states).grid(row=7, sticky=E, pady=4)
mainloop()
```

Pygame Module

We've had a brief look at the Pygame module already, but there's a lot more to it that needs to be explored. Pygame was developed to help Python programmers create games, whether they're graphical or not. You will, however, need to install Pygame before you can use it.

PYGAMING

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

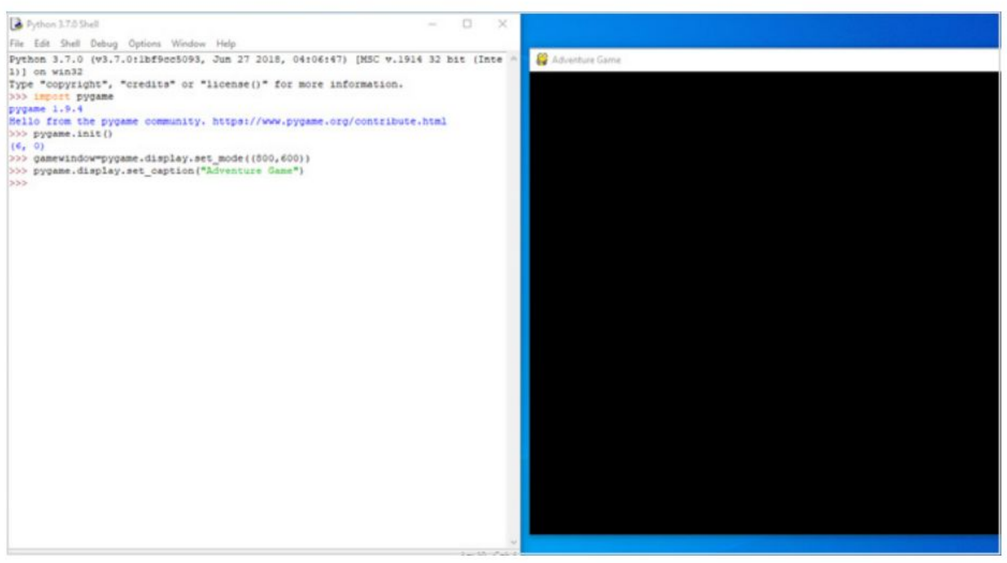
STEP 1 Naturally, we need to load up the Pygame modules into memory before we're able to utilise them. Once that's done, Pygame requires the user to initialise it prior to any of the functions being used:

```
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 will notice that after the first line is entered, you'll need to click back into the IDLE Shell to continue entering code. In addition, 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, we 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()
```

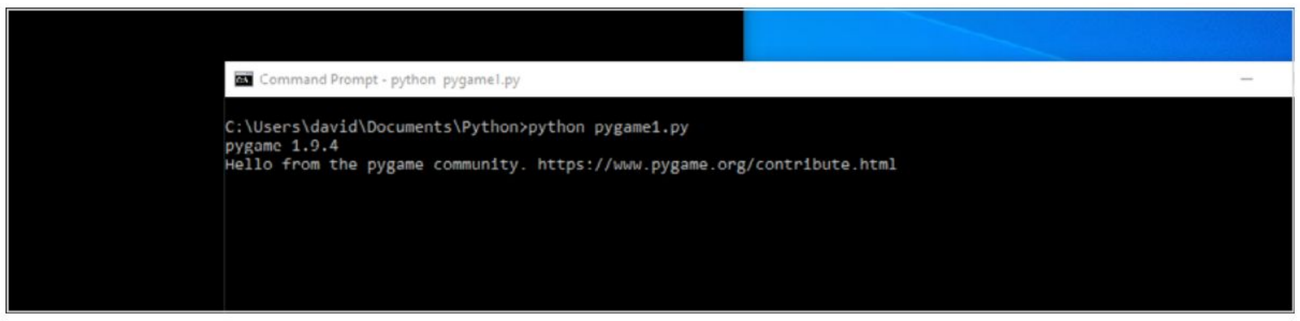
Untitled

File Edit Format Run Options Window Help

```
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

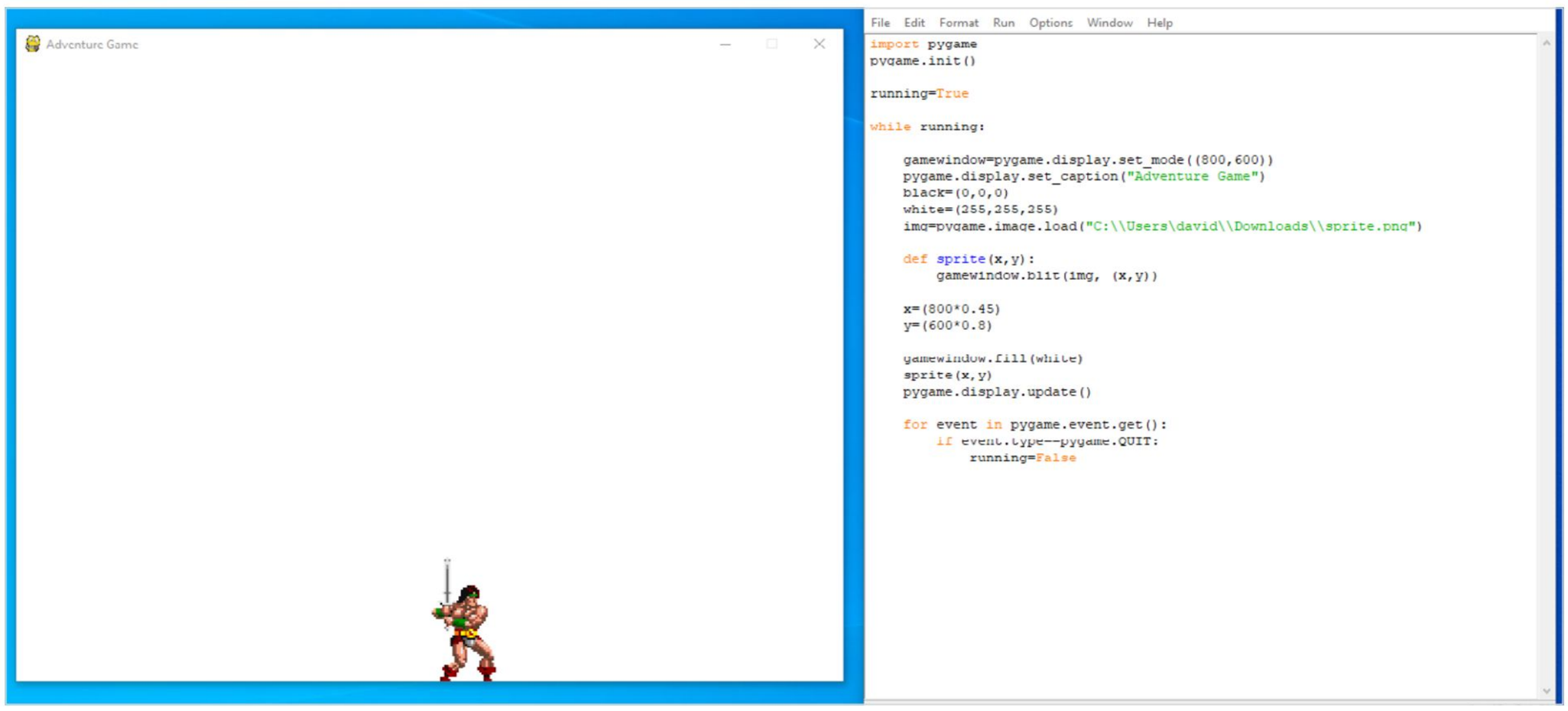
If the Pygame window still won't close, don't worry, it's just a discrepancy between the IDLE (which is written with Tkinter) and the Pygame module. If you run your code via the command line, it will close perfectly fine.



STEP 5

We'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. Also, 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("C:\\Users\david\\
Downloads\\sprite.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

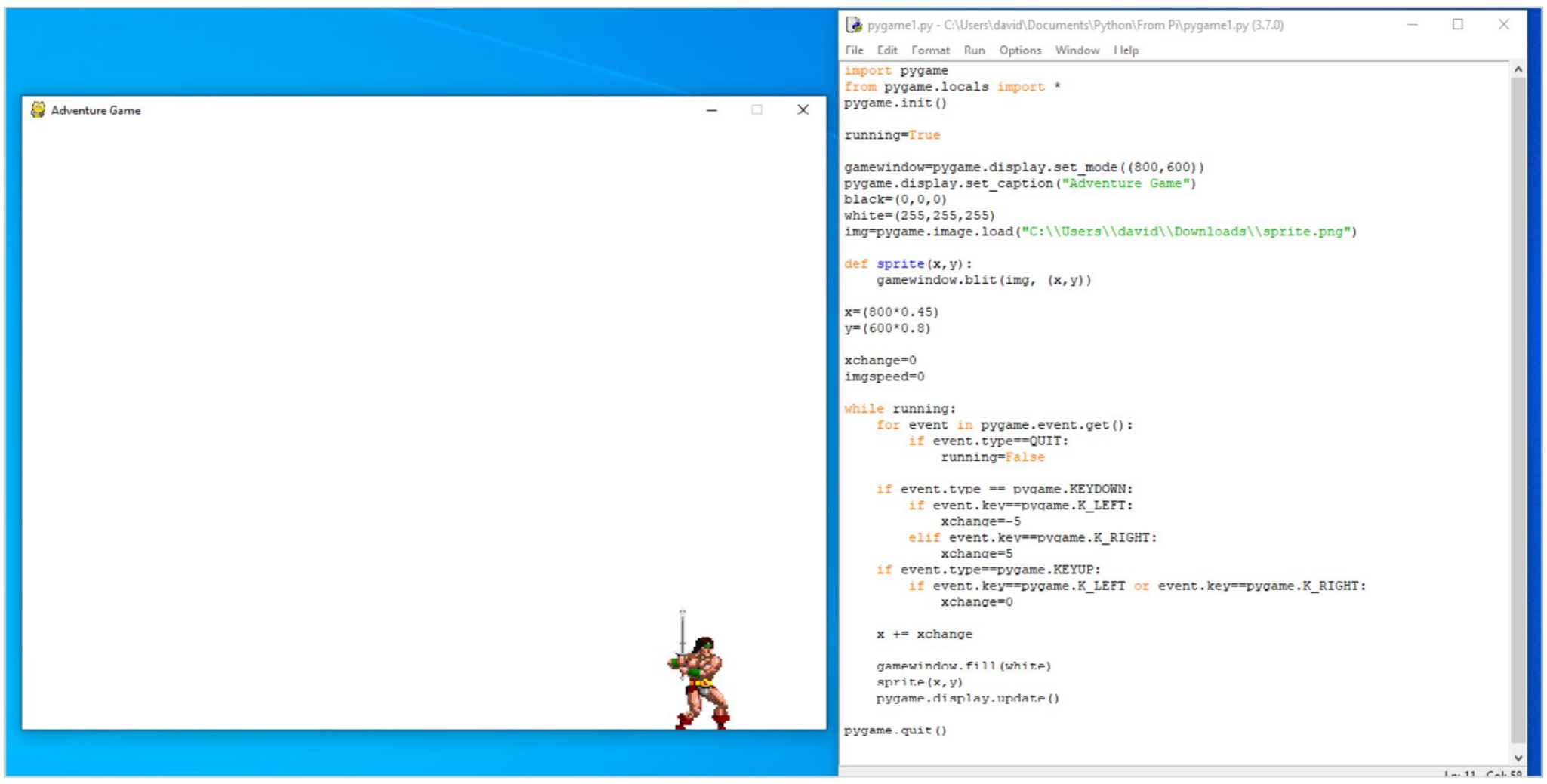
Let's quickly go through the code changes. We've defined two colours, black and white, together with their respective RGB colour values. Next, we've loaded the downloaded image called **sprite.png**, and allocated it to the variable img. We've also defined a sprite function, and the Blit function, which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

```
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("C:\\Users\david\\Downloads\\sprite.p

    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 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("C:\\Users\\david\\
Downloads\\sprite.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 will be able to move your sprite across the bottom of the screen. It looks like we have the makings of a classic arcade 2D scroller in the works.

```
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("C:\\Users\\david\\Downloads\\sprite.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 9 We can now implement a few additions, and utilise some previous tutorial code. The new elements are in 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(“python C:\\Users\\david\\
Documents\\Python\\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()
```

pygametxt.py - C:\Users\david\Documents\Python\From Pi\pygametxt.py (3.7.0)

File Edit Format Run Options Window Help

```
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("python C:\\Users\\david\\Documents\\Python\\pygame1.p


    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...")
```

STEP 10 When we 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 we can move the warrior sprite around. Overall, the effect is quite good, but there's always room for improvement.

Create Your Own Modules

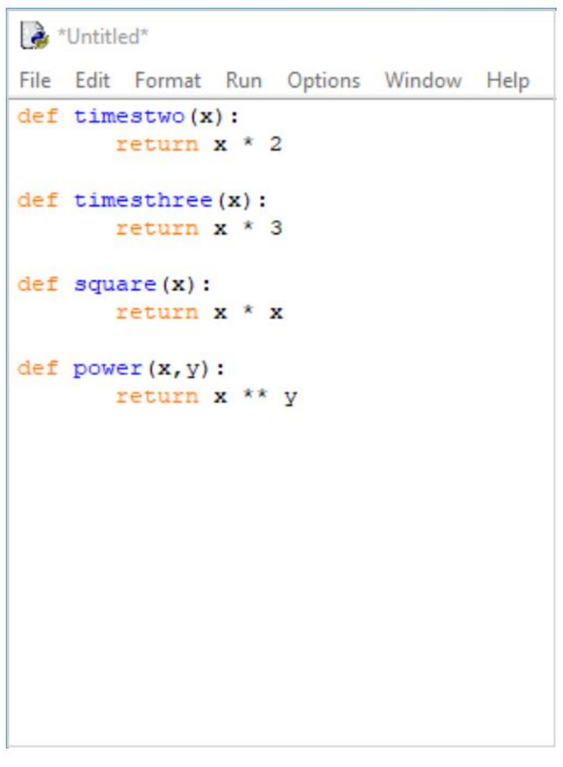
Large programs can be much easier to manage if you break them up into smaller parts and import the parts you need as modules. Learning to build your own modules also makes it easier to understand how modules work.

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 or three, and square or raise a number to an exponent (power). Create a New File in the IDLE and enter:

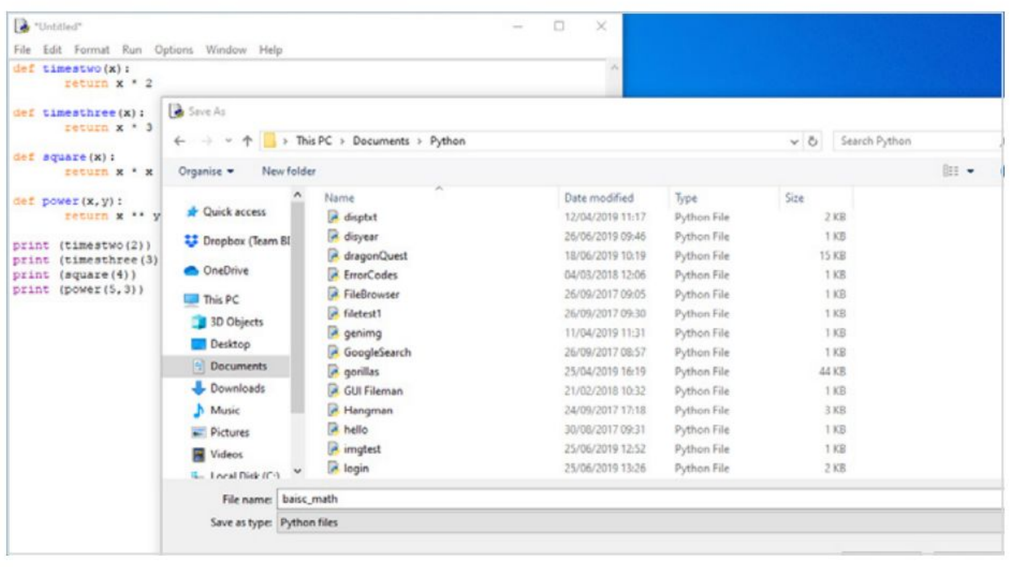
```
def timestwo(x):
        return x * 2

def timesthree(x):
        return x * 3

def square(x):
        return x * x

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



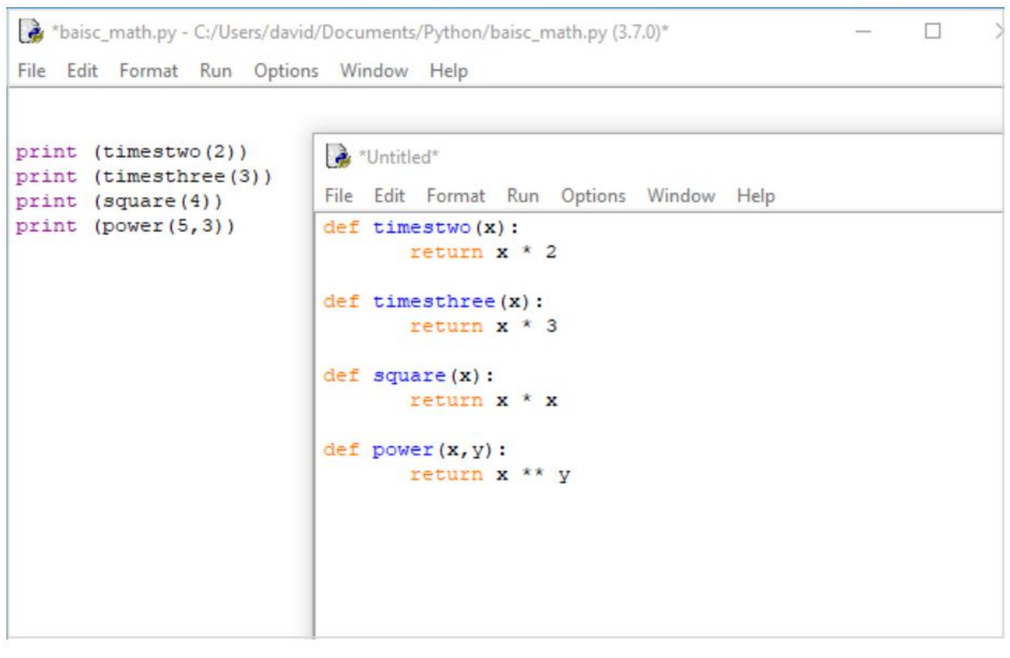
STEP 2 Under the above code, enter functions to call the code:

```
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.



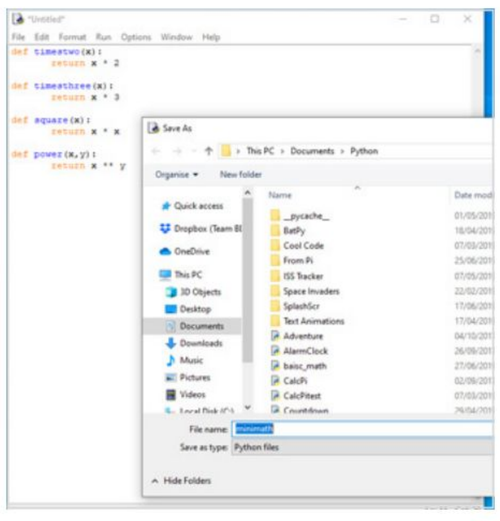
STEP 3 Now we're going to take the function definitions out of the program and into a separate file. Highlight the function definitions and choose **Edit > Cut**. Choose **File > New File** and use **Edit > Paste** in the new window. We now have two separate files, one with the function definitions, and 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 "C:/Users/david/Documents/Python/baisc_math.py", line 3, in <module>
    print (timestwo(2))
NameError: name 'timestwo' is not defined
>>>
```

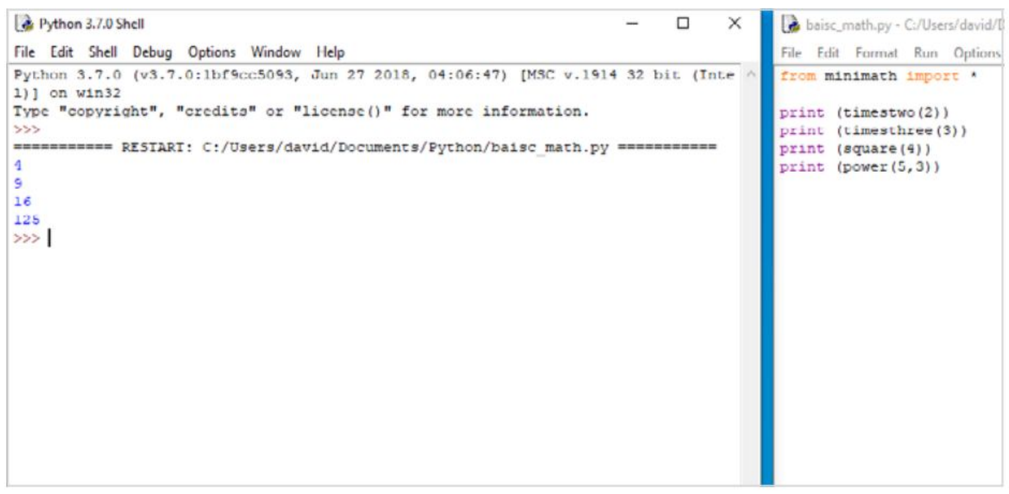
STEP 5 Return to the newly created window containing the function definitions, and click **File > Save As**. Name this **minimath.py** and save it in the same location as the original basic_math.py program. Now close the minimath.py window, so the basic_math.py window is left open.



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, and see it in action.



STEP 7 We can now make the program a little more advanced, by utilising the newly created module to its full. Let's include some user interaction. Start by creating a basic menu from which the user can choose:

```
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**.

testmath.py - C:/Users/david/Documents/Python/testmath.py (3.7.0)

File Edit Format Run Options Window Help

```
from minimath import *

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):")

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

STEP 9 Finally, we 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")
```

testmath.py - C:/Users/david/Documents/Python/testmath.py (3.7.0)

File Edit Format Run Options Window Help

```
from minimath import *

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):")

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

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.