Date and Time

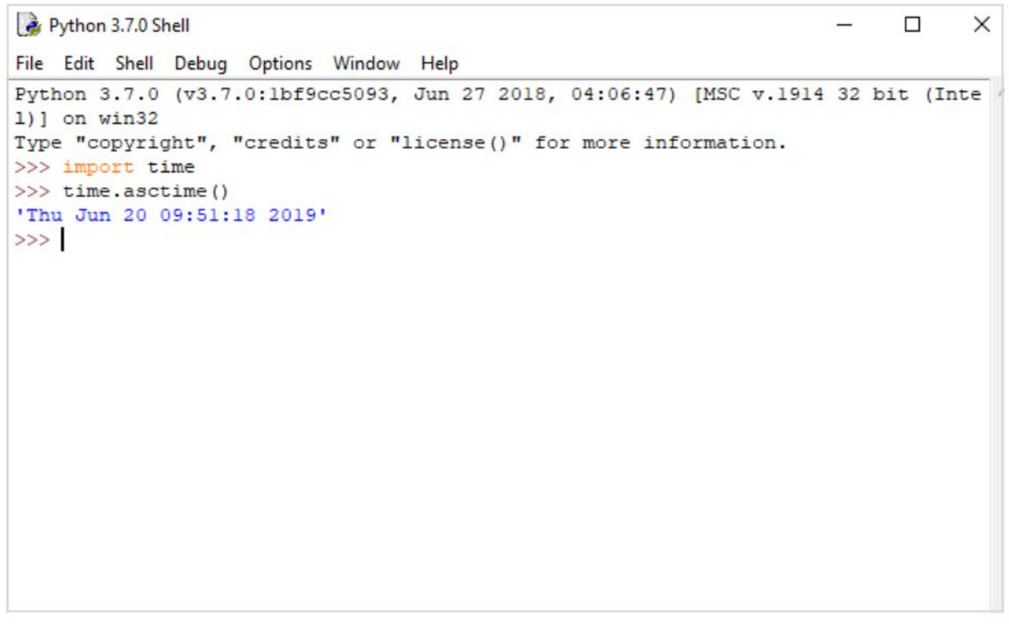
When working with data it's often handy to have access to the time. For example, you may want to time-stamp an entry, or see at what time a user logged into the system, and for how long. Thankfully, acquiring the time and date is easy thanks to the Time module.

TIME LORDS

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

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

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



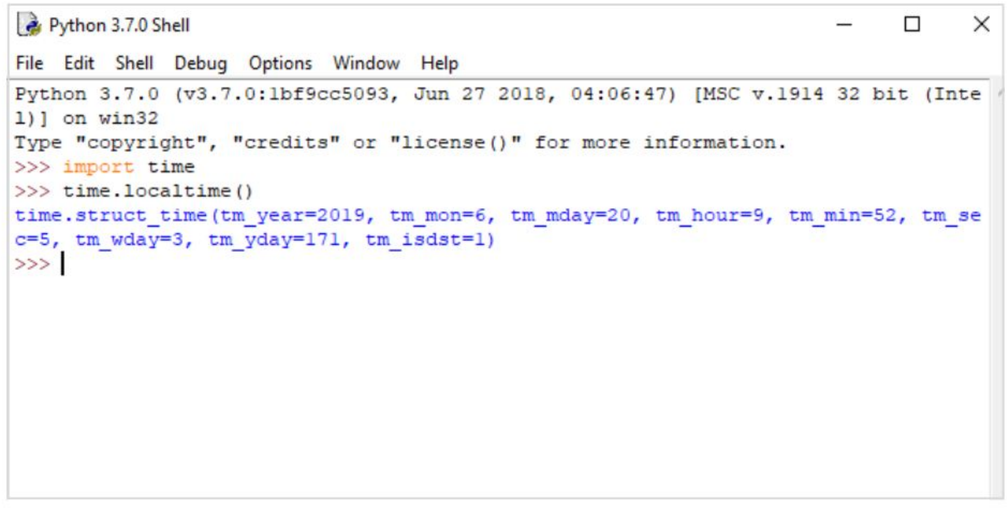
STEP 2 The time function is split into nine tuples, these are divided up into indexed items, as with any other tuple, and shown in the screen shot below.

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

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

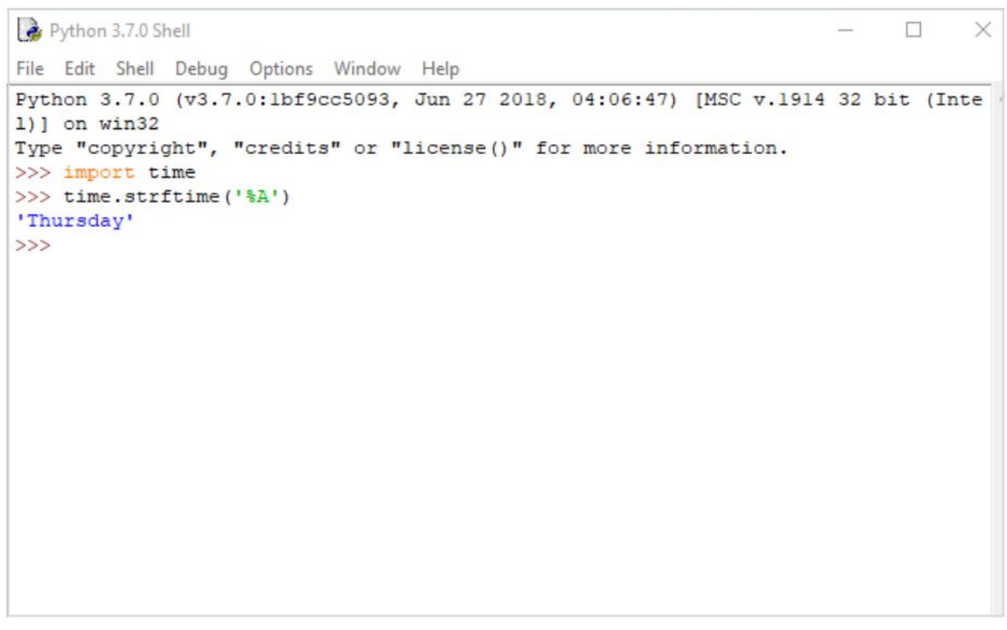
```
time.local.time()
```

The output is displayed as such: `'time.struct_time(tm_year=2019, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=13, tm_wday=3, tm_yday=250, tm_isdst=0)'`; obviously dependent on your current time, as opposed to the time this book was written.



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

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

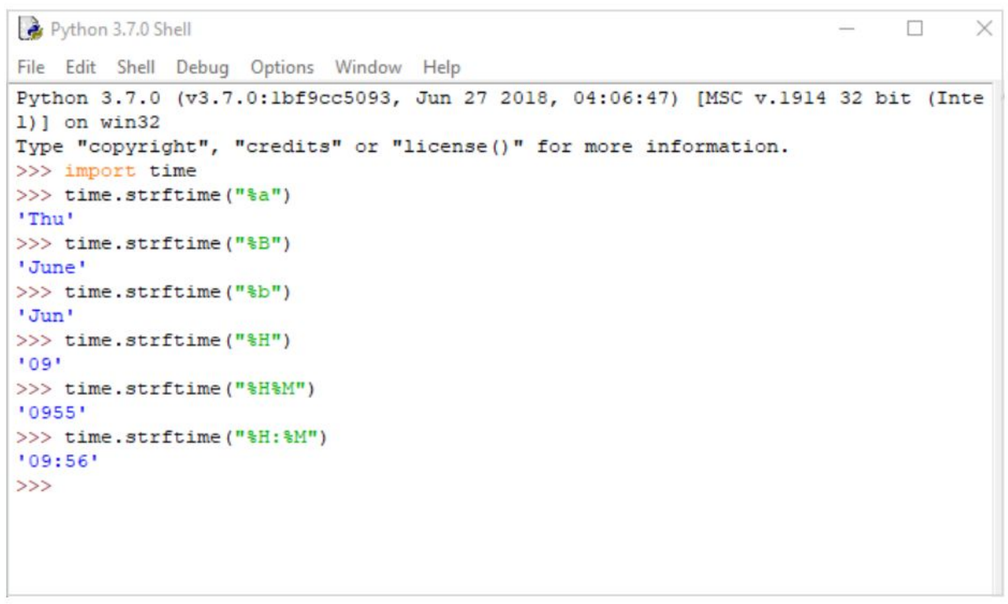


STEP 5 Naturally, this means you can incorporate various functions into your own code, such as:

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

STEP 6 Note the last two entries, with %H and %H%M, as you can see, these are the hours and minutes and as the last entry indicates, entering them as %H%M doesn't display the time correctly in the Shell. We can easily rectify this with:

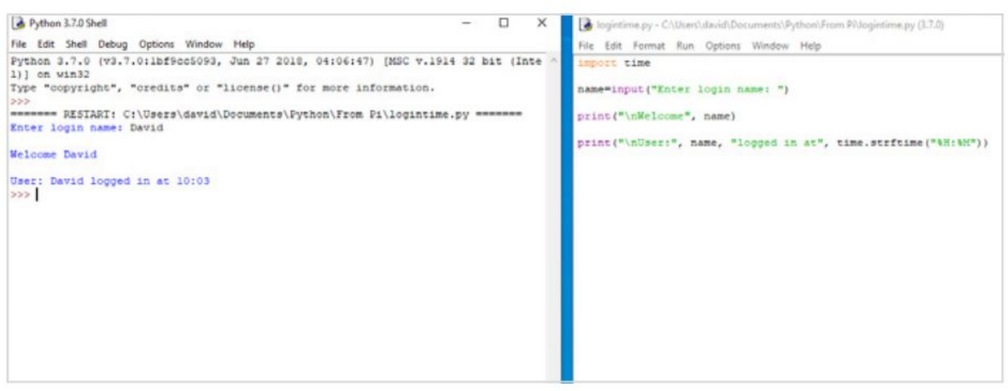
```
time.strftime("%H:%M")
```



STEP 7 This means you're going to be able to display either the current time, or the time when something occurred, such as a user entering their name. Try this code in the Editor:

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

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

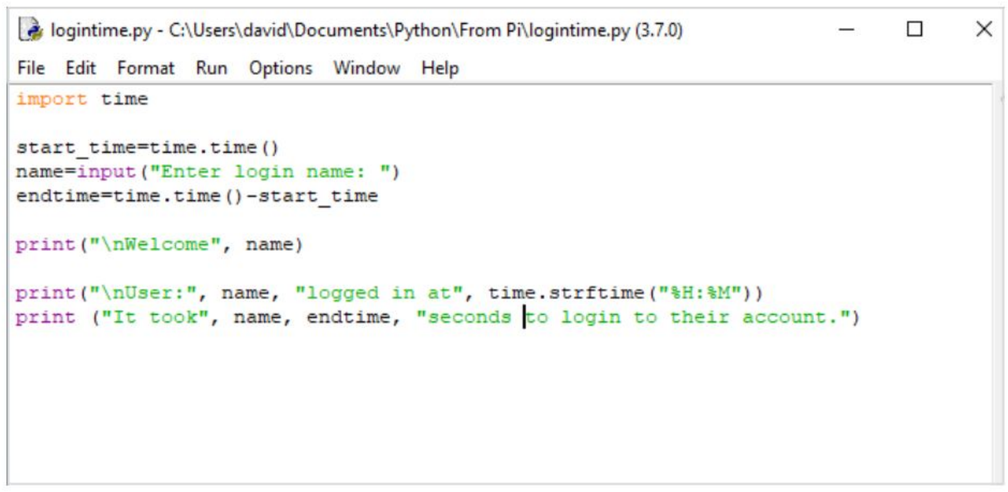


STEP 8 We can also use the Time module to display the amount of time taken for an event to happen. For example, taking the above code, we can alter it slightly by including:

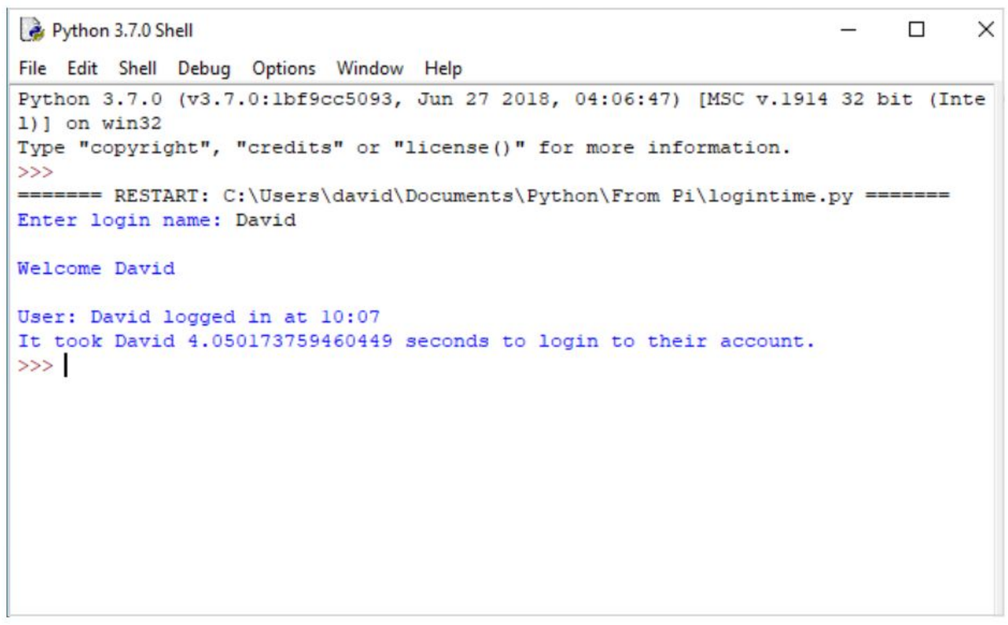
```
start_time=time.time()
```

And.

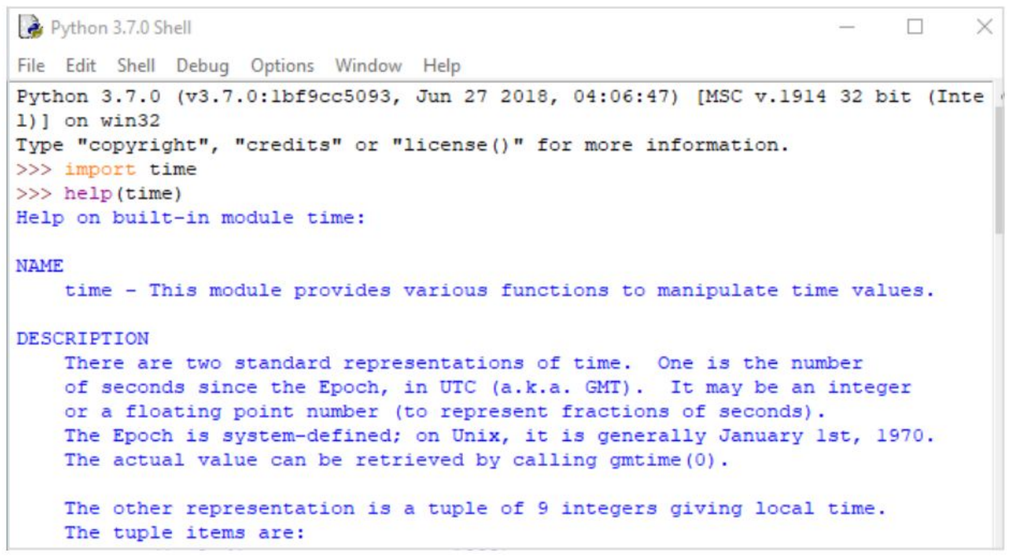
```
endtime=time.time()-start_time
```



STEP 9 The output will look similar to the screenshot below. The timer function needs to be either side of the input statement, as that's when the variable name is being created – depending on how long the user took to log in. The length of time is then displayed on the last line of the code, as the **endtime** variable.



STEP 10 There's a lot that can be done with the Time module, some of it is quite complex too – such as displaying the number of seconds since January 1st 1970. If you want to drill down further into the Time module, then in the Shell enter: `help(time)` to display the current Python version help file for the Time module.



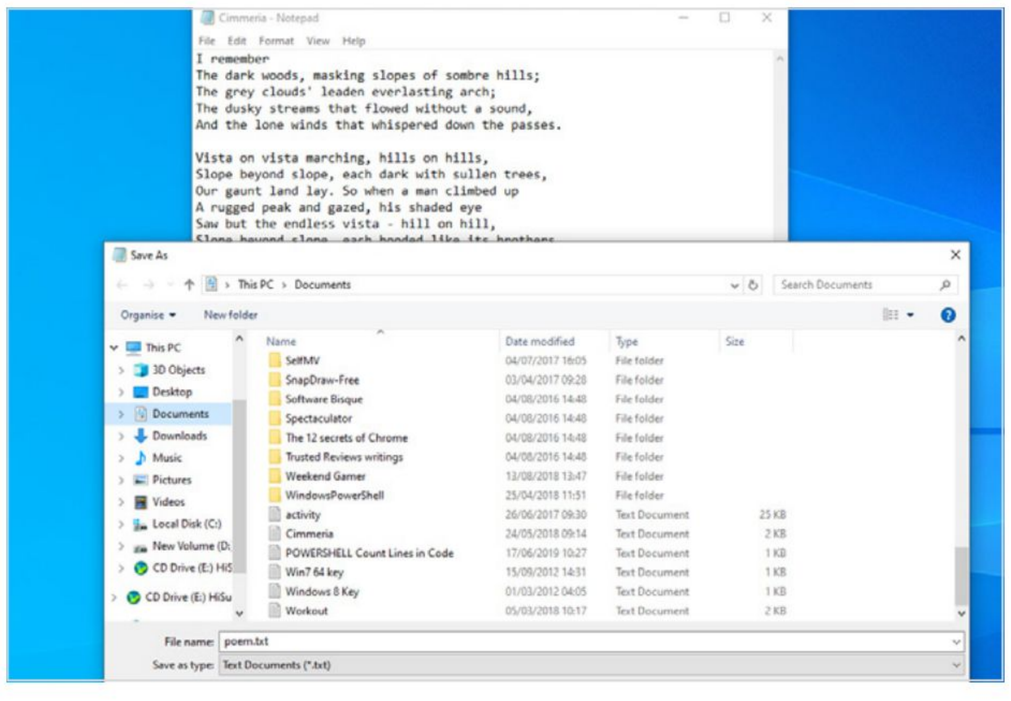
Opening Files

In Python, you can read text and binary files in your programs. This enables you to import data from one source to Python; handy if you have another program language running, that's creating an output, and you want to analyse the results in Python.

OPEN, READ AND WRITE

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

STEP 1 Start by entering some text into your system's text editor. The text editor is preferable to a word processor, as word processors include background formatting and other elements. In our example, we have the poem The Cimmerian, by Robert E Howard, and we've saved the file as **poem.txt**.



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

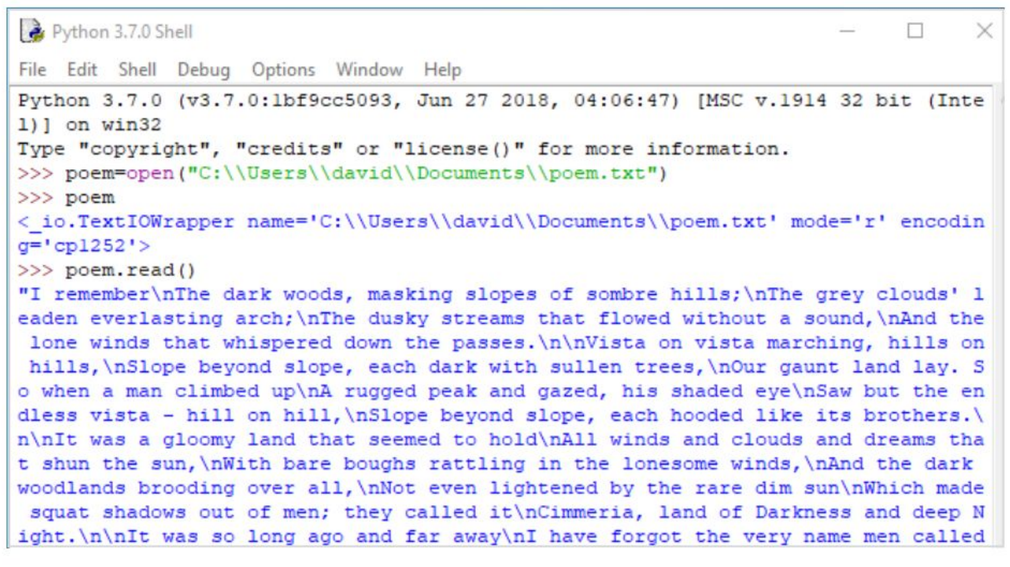
```
poem=open("C:\\Users\\david\\Documents\\poem.txt")
```

The reason for the double slash (\\) is because Python will read this as a Unicode Error, thinking you've entered: \U. This is Windows-only, Linux and Mac won't have this issue.

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

```
poem.read()
```

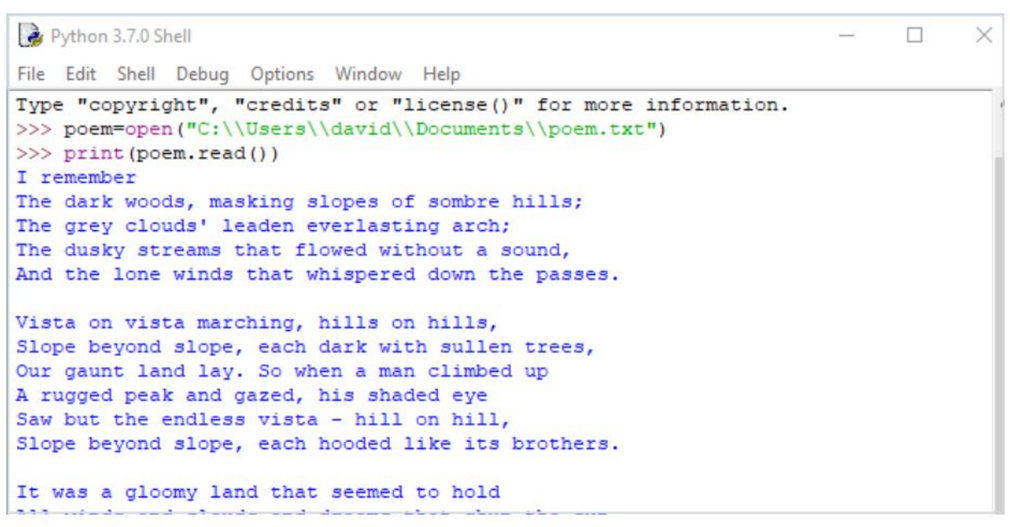
Note than a **/n** entry in the text represents a new line, as we have used previously.



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

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

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



STEP 5

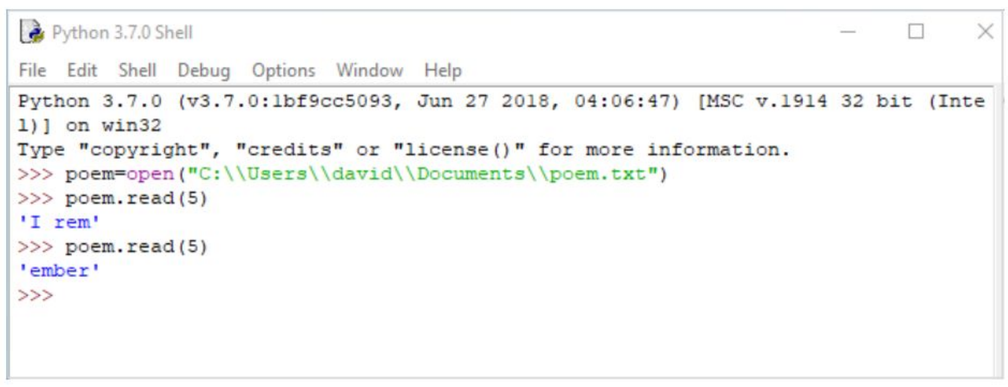
As with lists, tuples, dictionaries and so on, you're able to index individual characters of the text. For example:

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

Displays the first five characters, while entering:

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

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



STEP 6

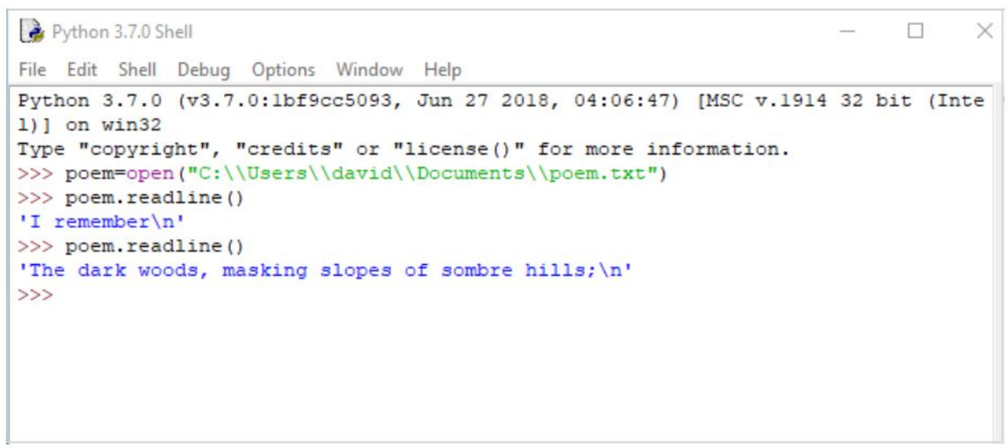
Similarly, you can display one line of text at a time by using the **readline()** function. For example:

```
poem=open("C:\\Users\\david\\Documents\\poem.txt")
poem.readline()
```

Will display the first line of the text. And:

```
poem.readline()
```

Will display the next line of text.



STEP 7

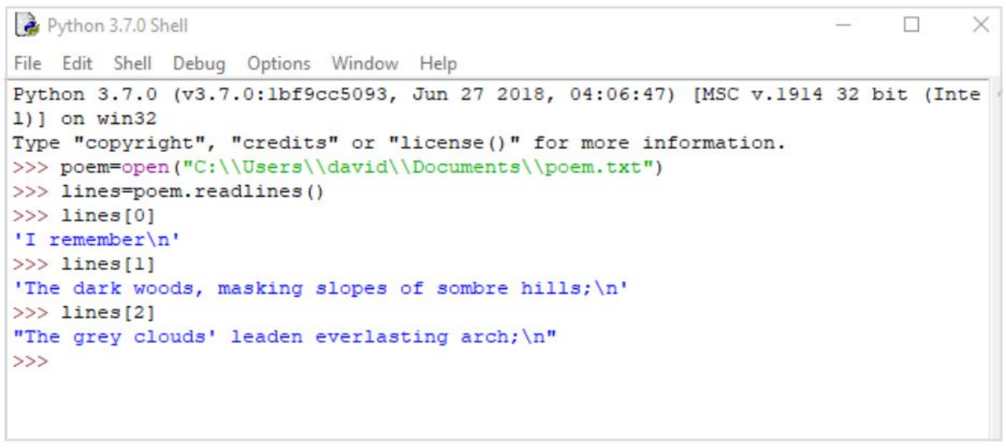
As you may suspect, you can pass the **readline()** function into a variable, allowing you to call it again, when needed:

```
poem=open("C:\\Users\\david\\Documents\\poem.txt")
line=poem.readline()
line
```

STEP 8

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

```
poem=open("C:\\Users\\david\\Documents\\poem.txt")
lines=poem.readlines()
lines[0]
lines[1]
lines[2]
```



STEP 9

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

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

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

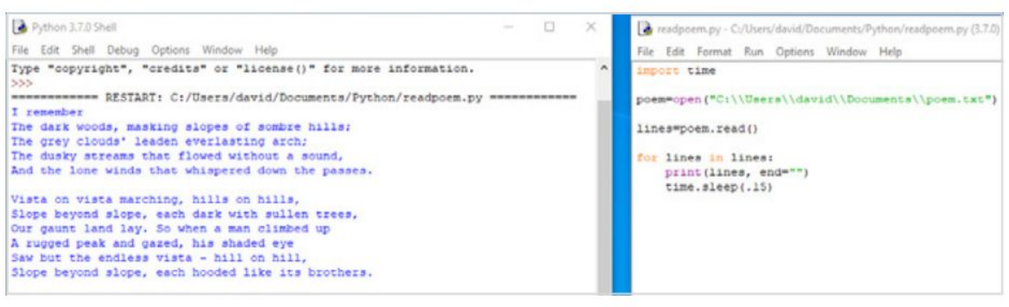
```
poem=open("C:\\Users\\david\\Documents\\poem.txt")
for lines in poem:
        print(lines)
```

STEP 10

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

```
import time
poem=open("C:\\Users\\david\\Documents\\poem.txt")
lines=poem.read()
for lines in lines:
    print(lines, end="")
    time.sleep(.15)
```

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



Writing to Files

Being able to read external files within Python is certainly handy, but writing to a file can be even more useful. Using the write() function, you're able to output the results of a program to a file, which you can then use to read() back into Python, or as a text file for perusal later.

WRITE AND CLOSE

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

STEP 1 Start by opening IDLE and enter the following (obviously entering your own username location):

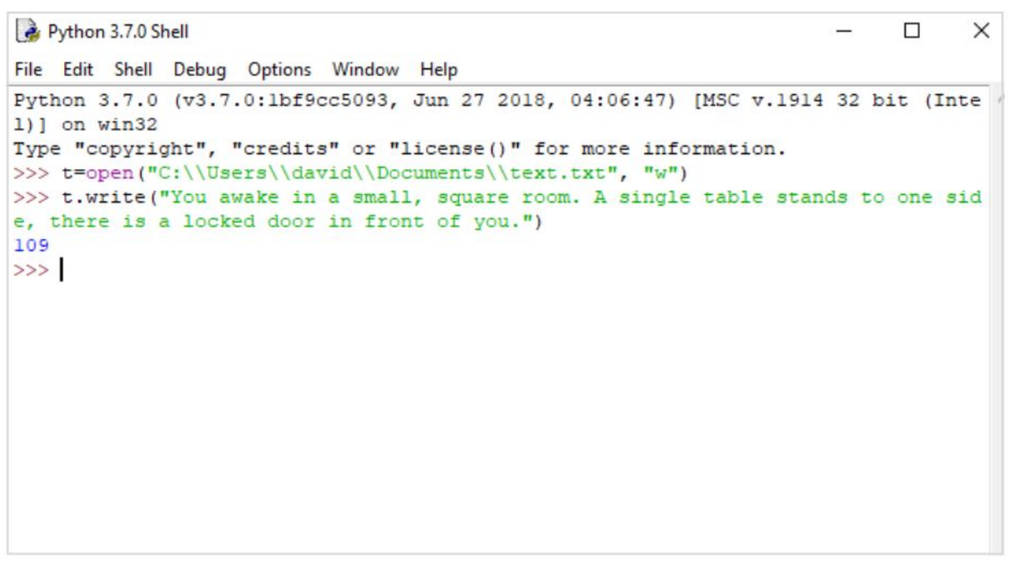
```
t=open("C:\\Users\\david\\Documents\\text.txt", "w")
```

This code will create a text file, called text.txt in write mode, using the variable 't'. If there's no file of that name in the location, it will create one. If one already exits, it will overwrite it – so be careful.

STEP 2 We can now write to the text file using the write() function. This works opposite to read(), writing lines instead of reading them. Try this:

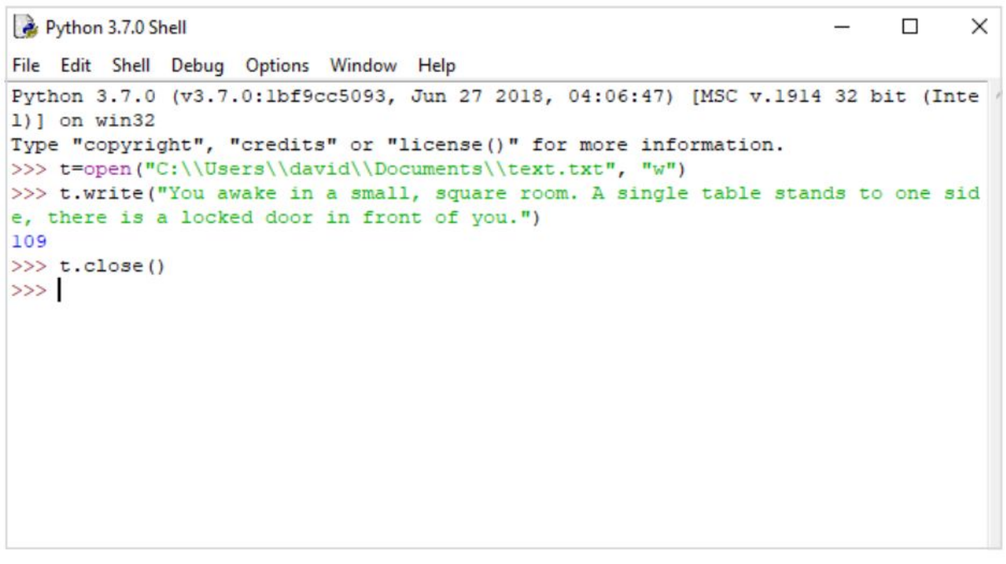
```
t.write("You awake in a small, square room. A single table stands to one side, there is a locked door in front of you.")
```

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

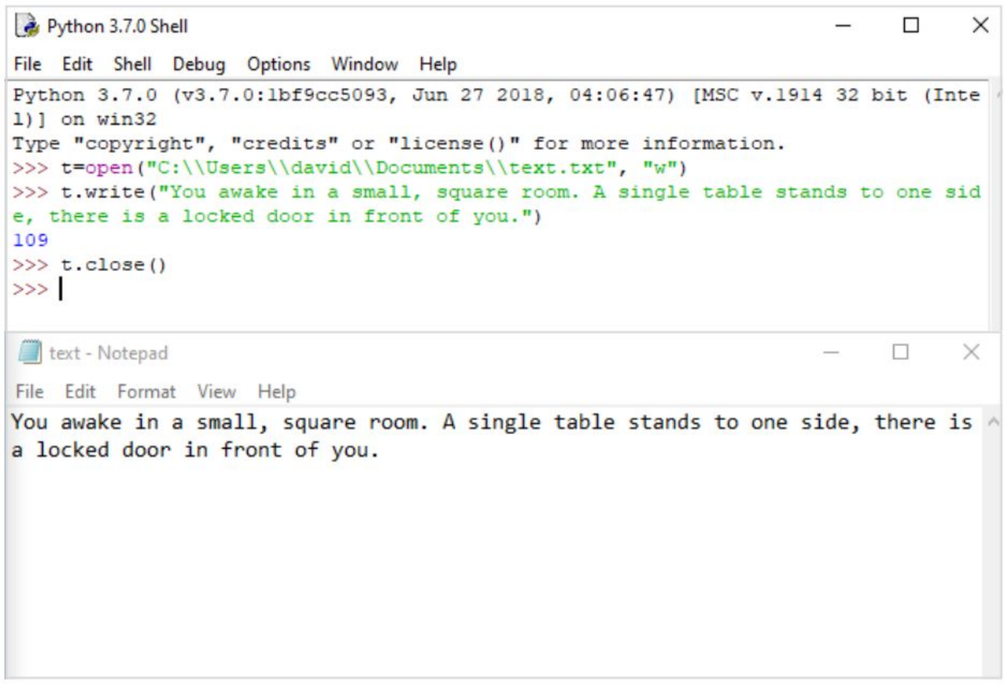


STEP 3 However, the actual text file is still blank (you can check by opening it up). This is because you've written the line of text to the file object, but not committed it to the file itself. Part of the write() function is that we need to commit the changes to the file, we can do this by entering:

```
t.close()
```



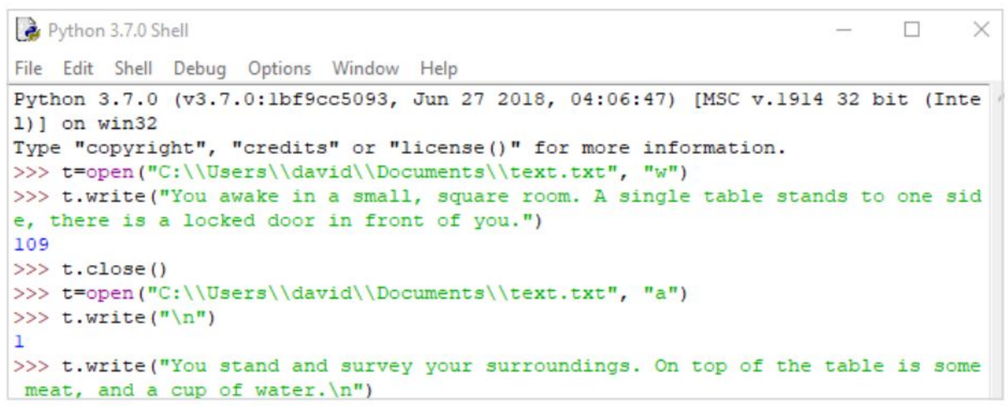
STEP 4 If you now open the text file with a text editor, you'll see that the line you created has been written to the file. This gives us the foundation for some interesting possibilities, perhaps the creation of your own log file, or even the beginning of an adventure game.



STEP 5

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

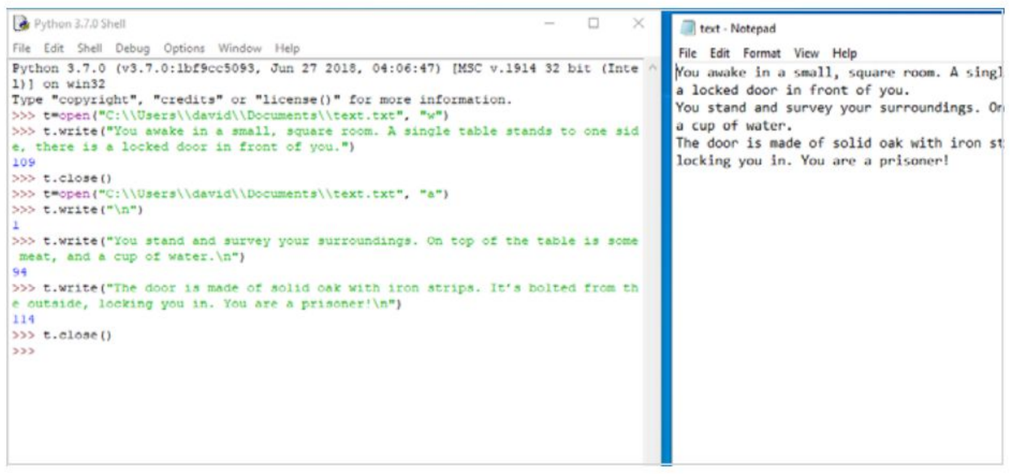
```
t=open("/home/pi/Documents/text.txt","a")
t.write("\n")
t.write(" You stand and survey your surroundings. On top of the table is some meat, and a cup of water.\n")
```



STEP 6

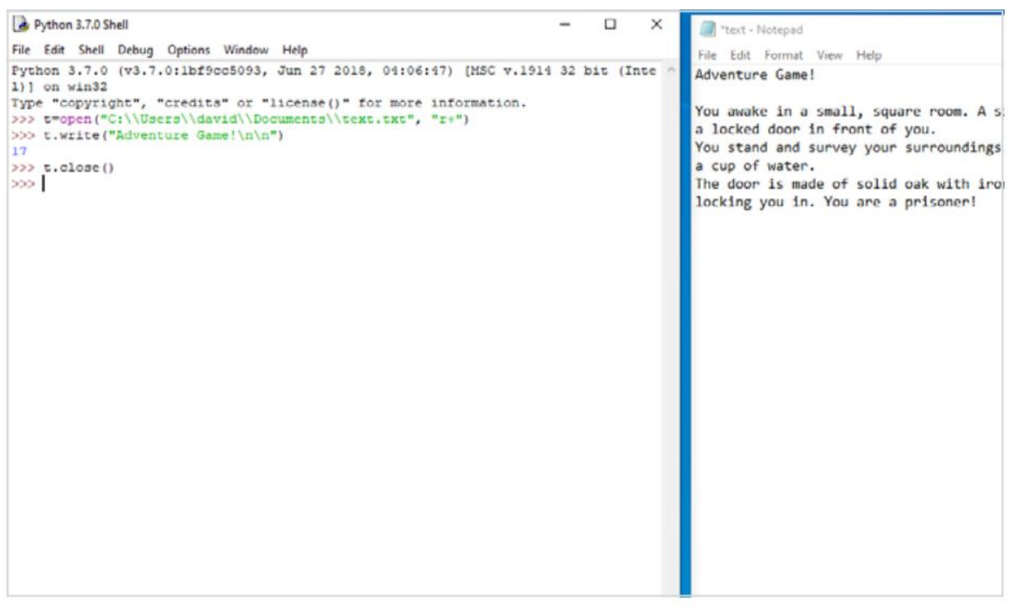
We can keep extending the text line by line, ending each with a new line (\n). When you're done, finish the code with t.close(), and open the file in a text editor to see the results:

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



STEP 7

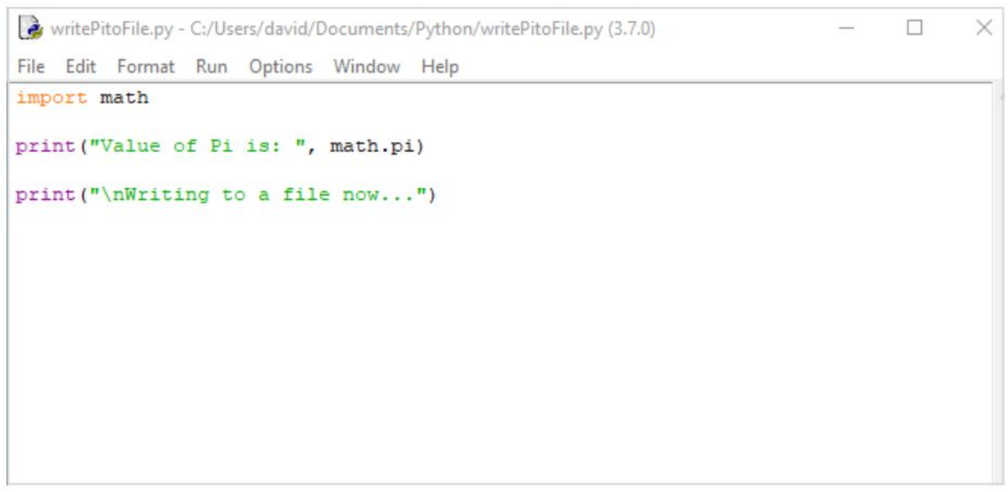
There are various types of file access to consider using the open() function. Each depends on how the file is accessed, and even the position of the cursor. For example, r+ opens a file in read and write, and places the cursor at the start of the file.



STEP 8

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

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



STEP 9

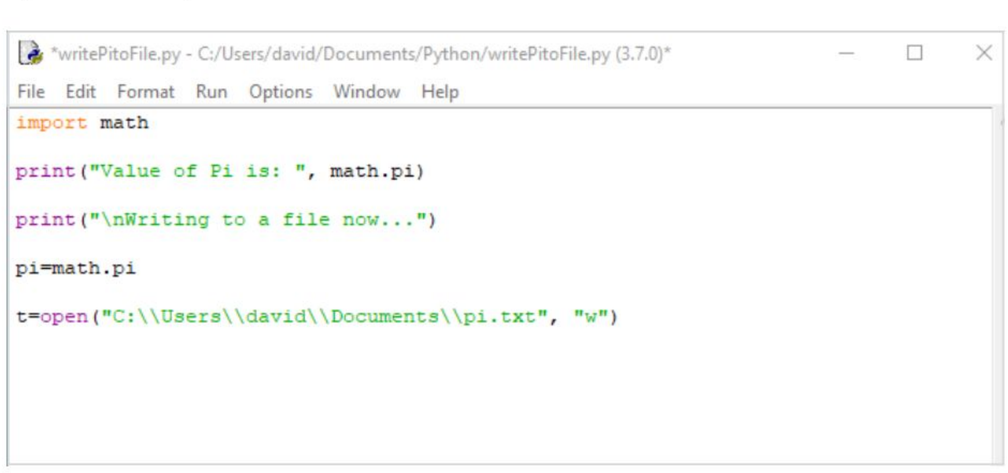
Now let's create a variable called pi, and assign it the value of Pi:

```
pi=math.pi
```

We also need to create a new file to write Pi to:

```
t=open("C:\\Users\\david\\Documents\\pi.txt", "w")
```

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

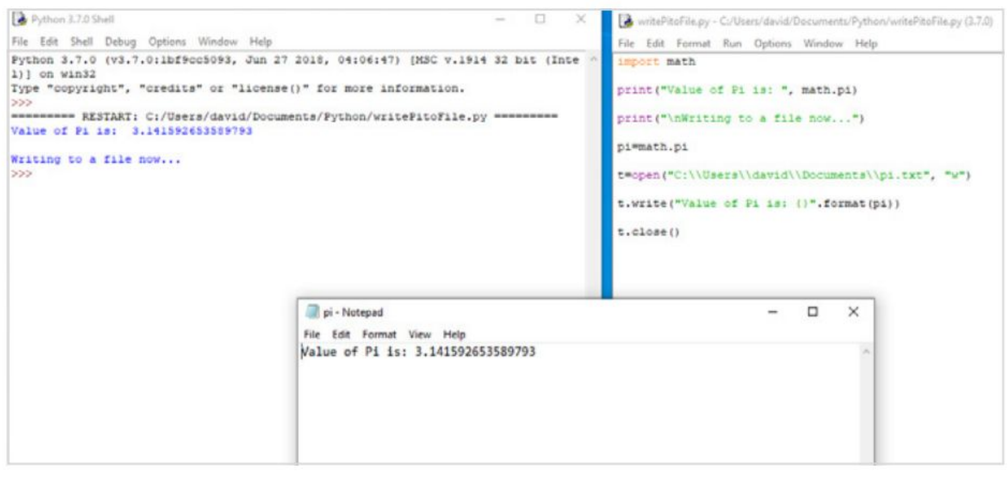


STEP 10

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

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

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



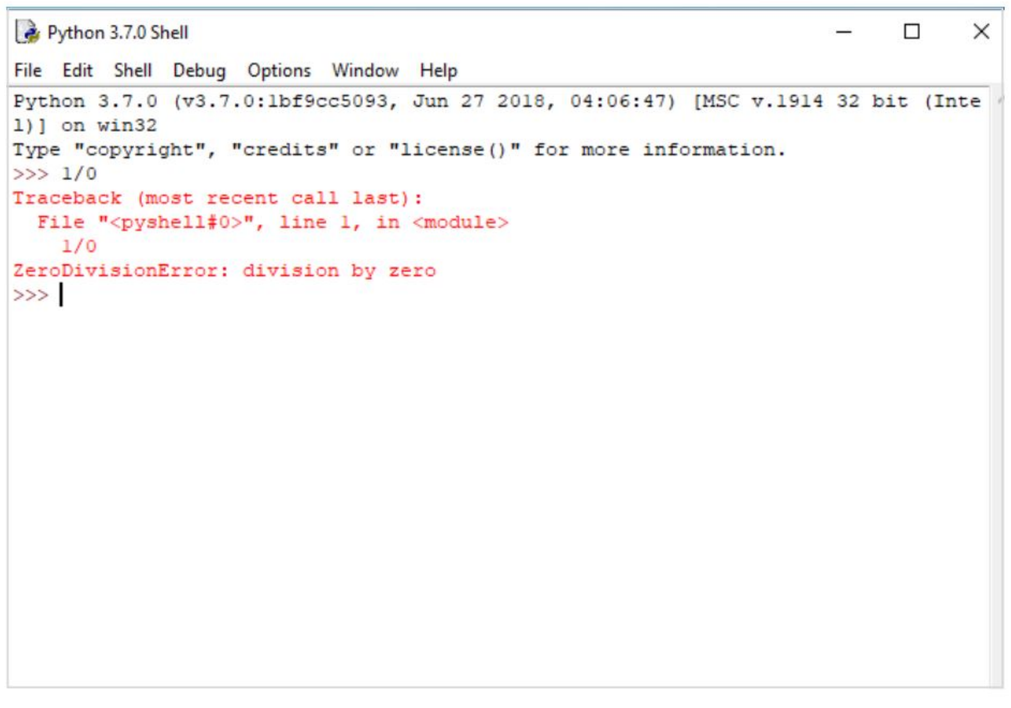
Exceptions

As your code begins to form and lengthen, you'll naturally come across some exceptional circumstances that are mostly out of your control. Let's assume you ask a user to divide two numbers, and they try to divide by zero. This will create an error, and break your code.

EXCEPTIONAL OBJECTS

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

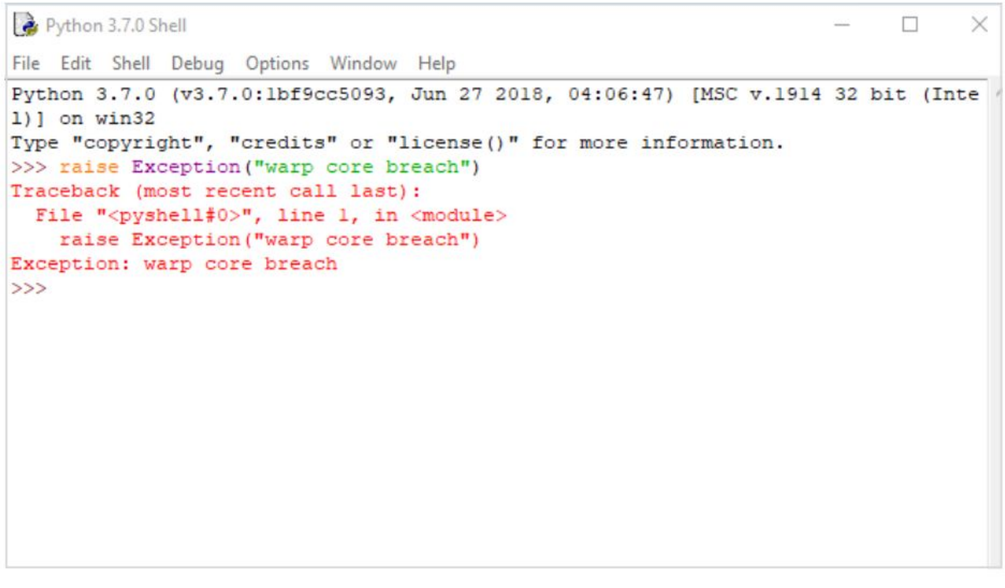
STEP 1 You can create an exception error by simply trying to divide a number by zero. This will report back with the **ZeroDivisionError: Division by zero** message, as seen in the screenshot. The ZeroDivisionError part is the exception class, of which there are many.



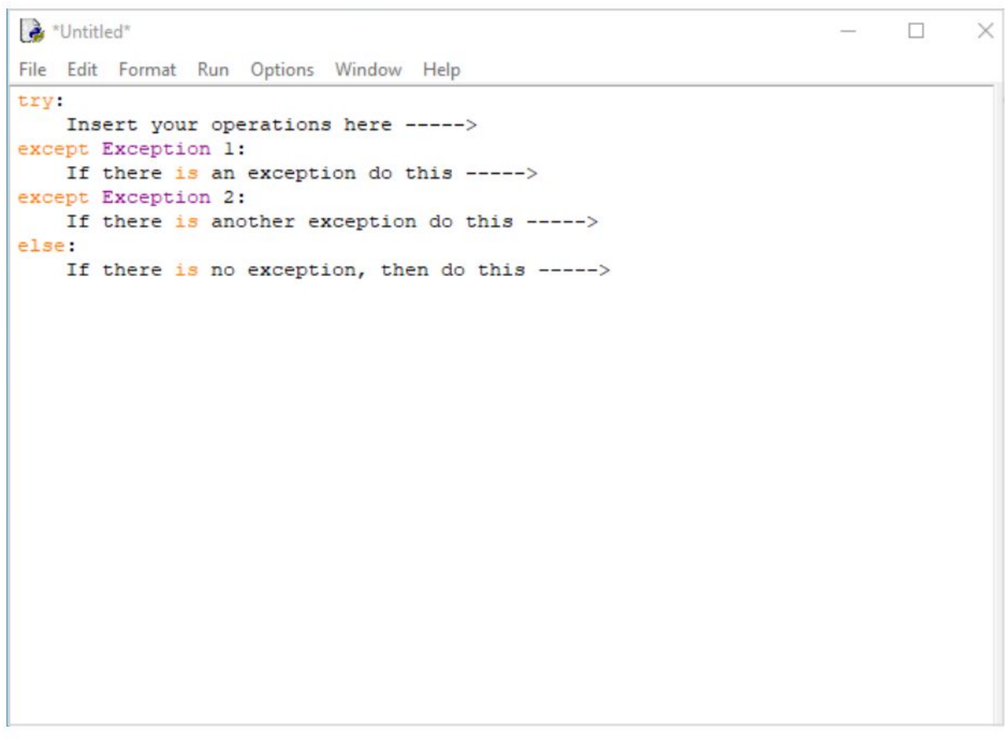
STEP 2 Most exceptions are raised automatically when Python comes across something that's inherently wrong with the code. However, we can create our own exceptions that are designed to contain the potential error and react to it, as opposed to letting the code fail.

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

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

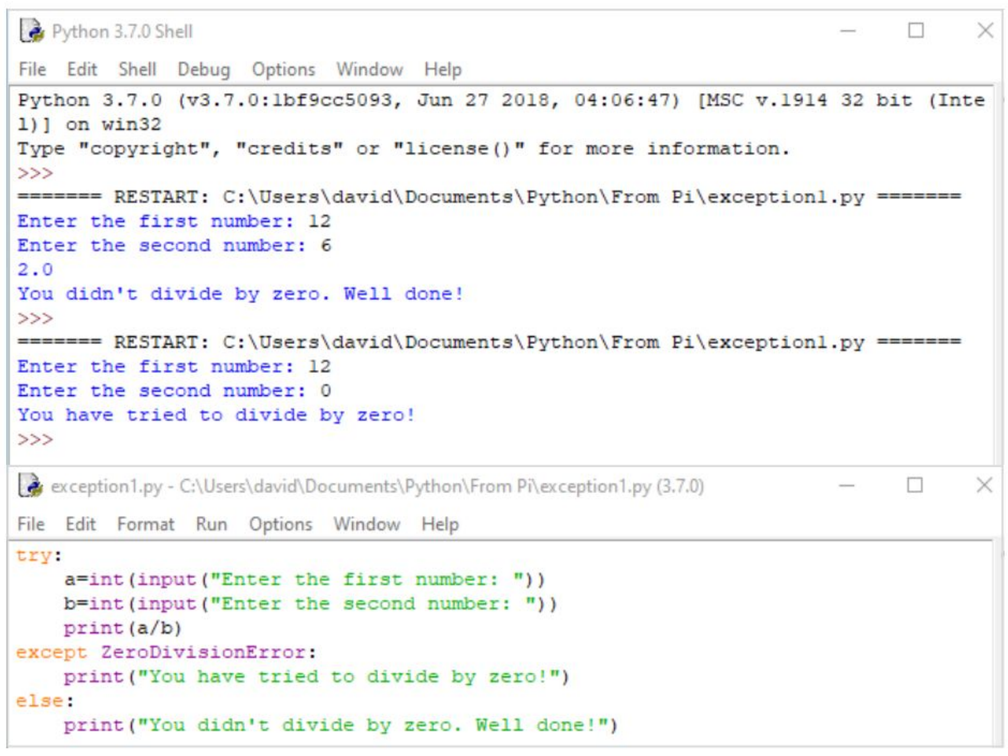


STEP 4 To trap any errors in the code we can encase the potential error within a **try**: block. This block consists of: try, except, else, where the code is held within try, then if there's an exception do something, or do something else.



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

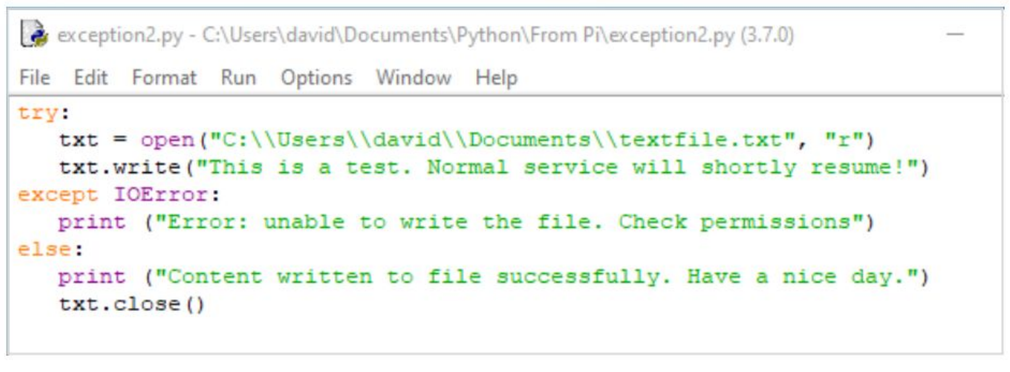
```
try:
 a=int(input("Enter the first number: "))
 b=int(input("Enter the second number: "))
 print(a/b)
except ZeroDivisionError:
 print("You have tried to divide by zero!")
else:
 print("You didn't divide by zero. Well done!")
```



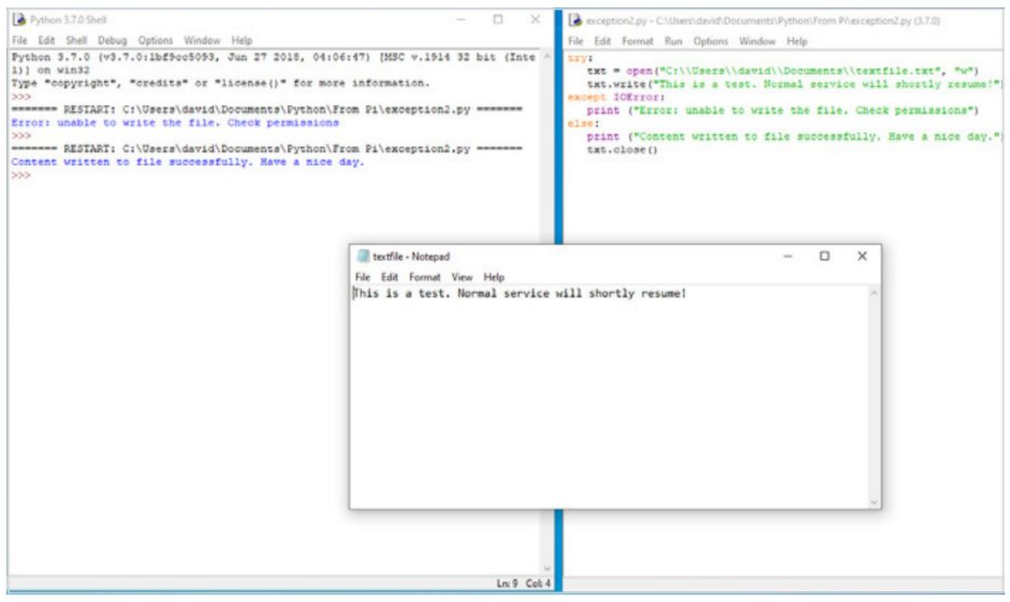
STEP 6 You can use exceptions to handle a variety of useful tasks. Using an example from our previous tutorials, let's assume you want to open a file and write to it:

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

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



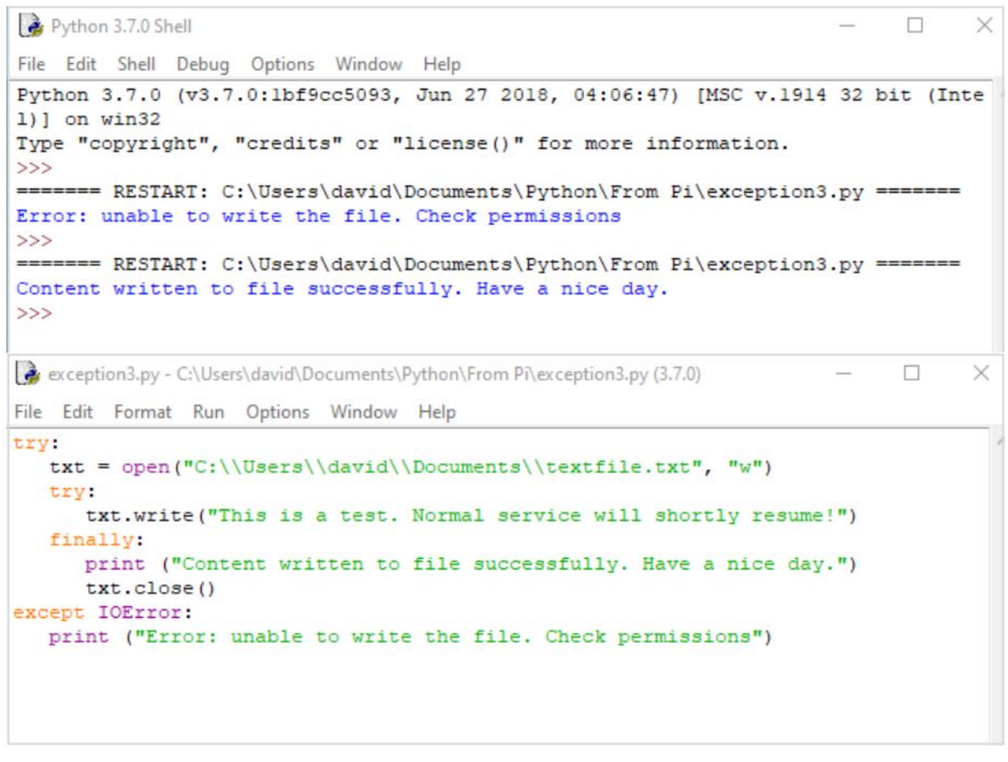
STEP 8 Naturally, we can quickly fix the issue by changing the "r" read only instance with a "w" for write. This, as you already know, will create the file and write the content then commit the changes to the file. The end result will report a different set of circumstances, in this case, a successful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



STEP 9 You can also use a **finally:** block, which works in a similar fashion, but you cannot use else with it. Hint: You'll need to delete the textfile.txt file from your folder.

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

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



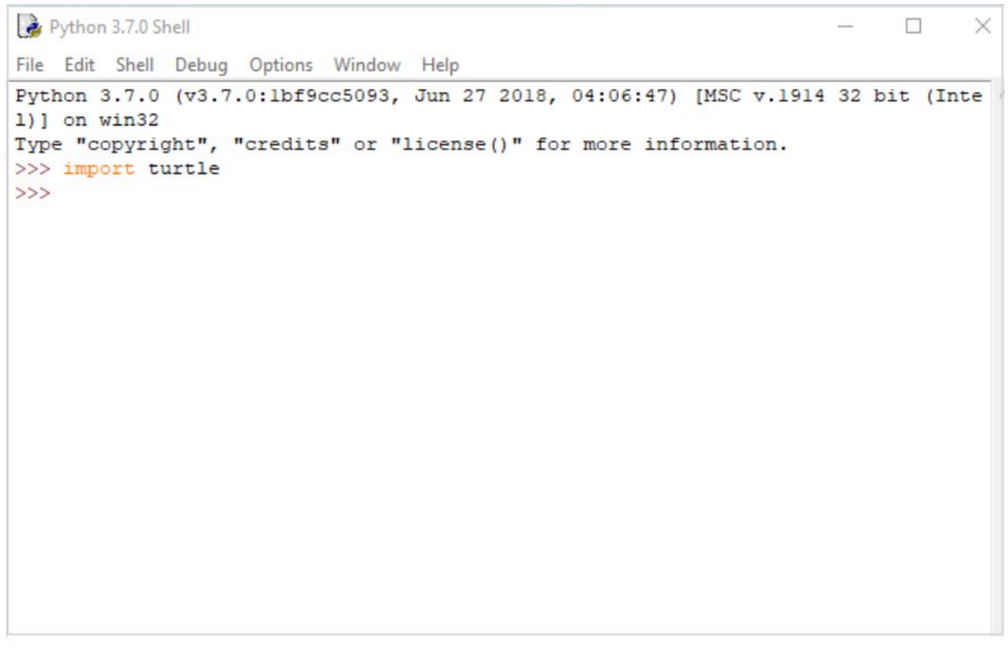
Python Graphics

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

GOING GRAPHICAL

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

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

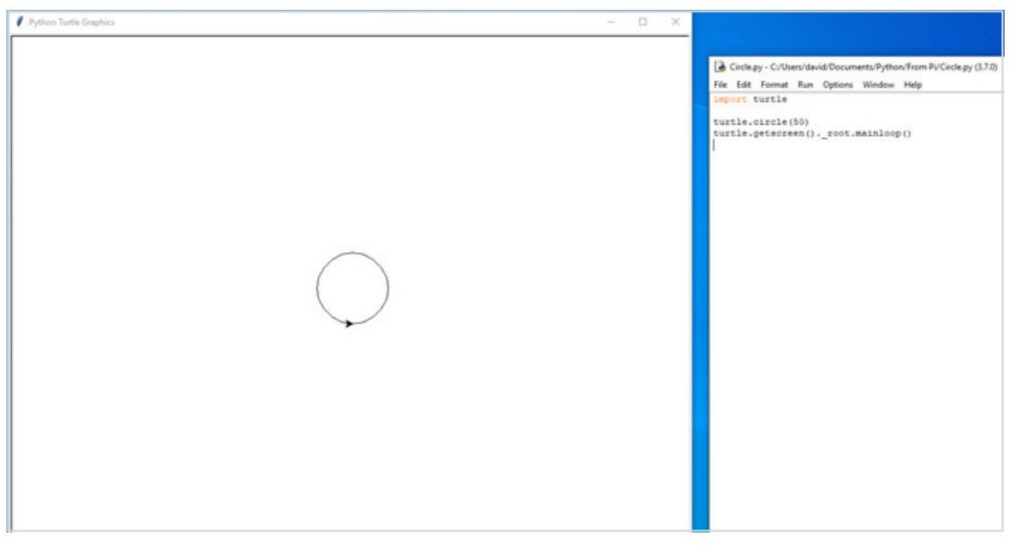


STEP 2 Let's begin by drawing a simple circle. Start a New File, then enter the following code:

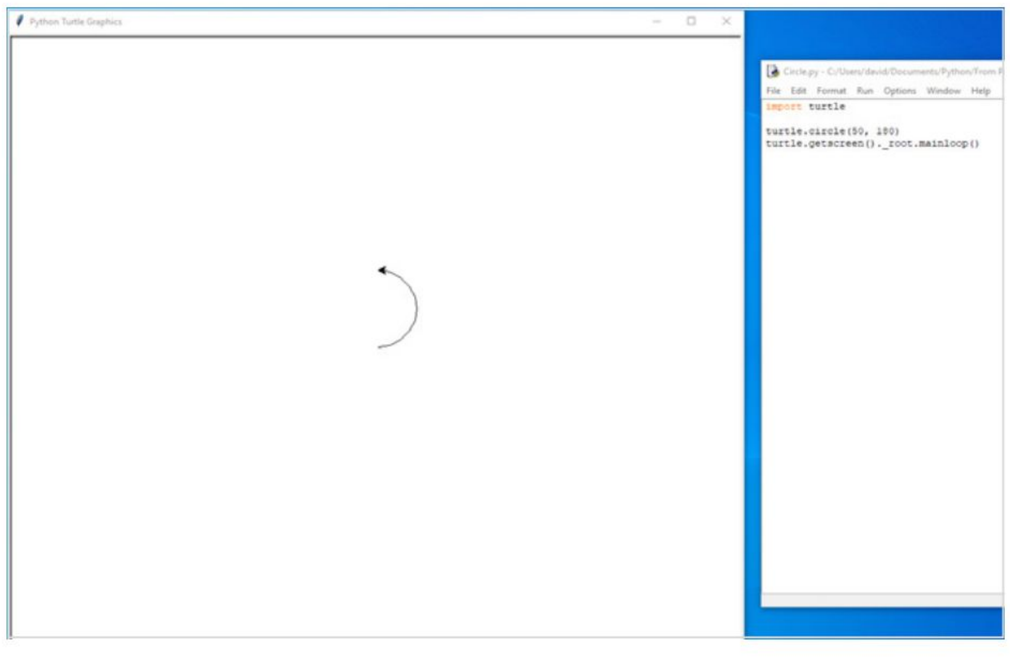
```
import turtle

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

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



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



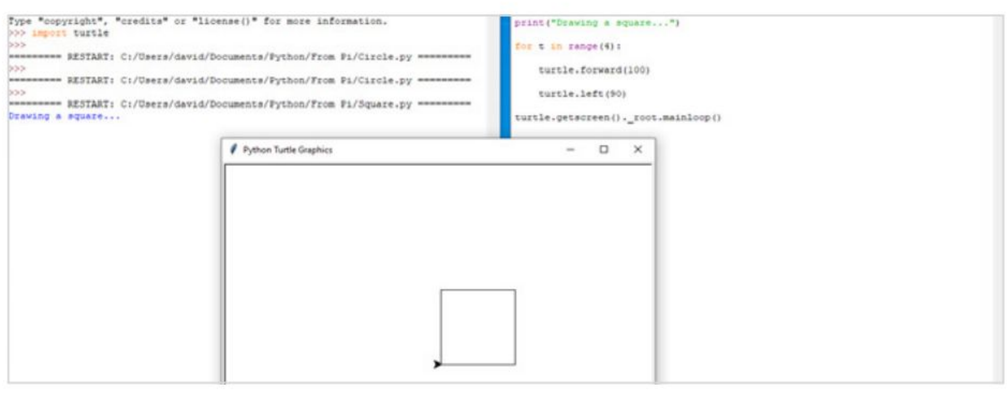
STEP 4 The last part of the circle code tells Python to keep the window where the drawing is taking place to remain open, so the user can click to close it. Now let's make a square:

```
import turtle

print("Drawing a square…")

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

You'll notice we've inserted a loop to draw the sides of the square.



STEP 5

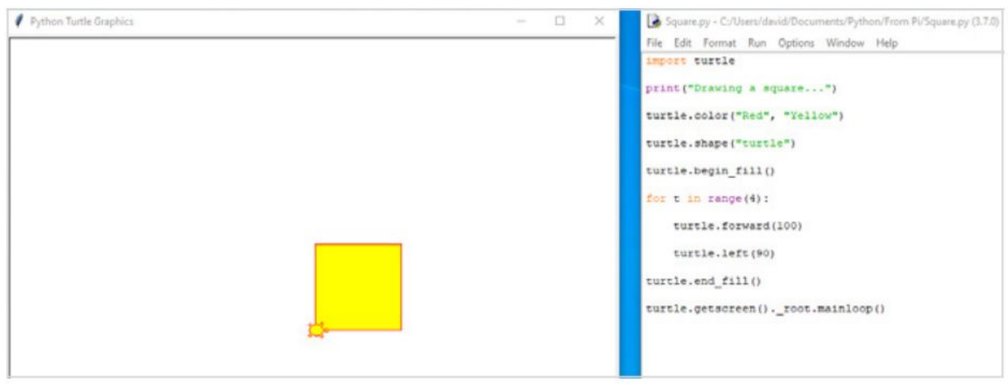
To add some colour, we can add a new line to the square code:

```
turtle.color(“Red”)
```

And we can even change the character to an actual turtle by entering:

```
turtle.shape(“turtle”)
```

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

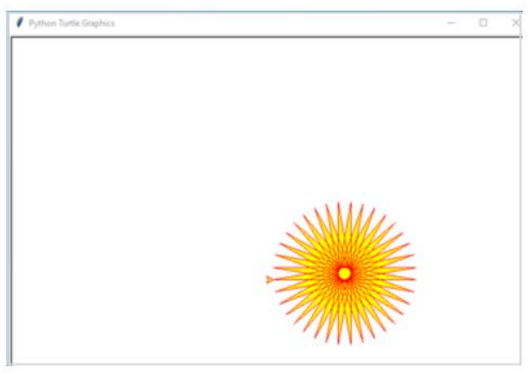


STEP 6

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

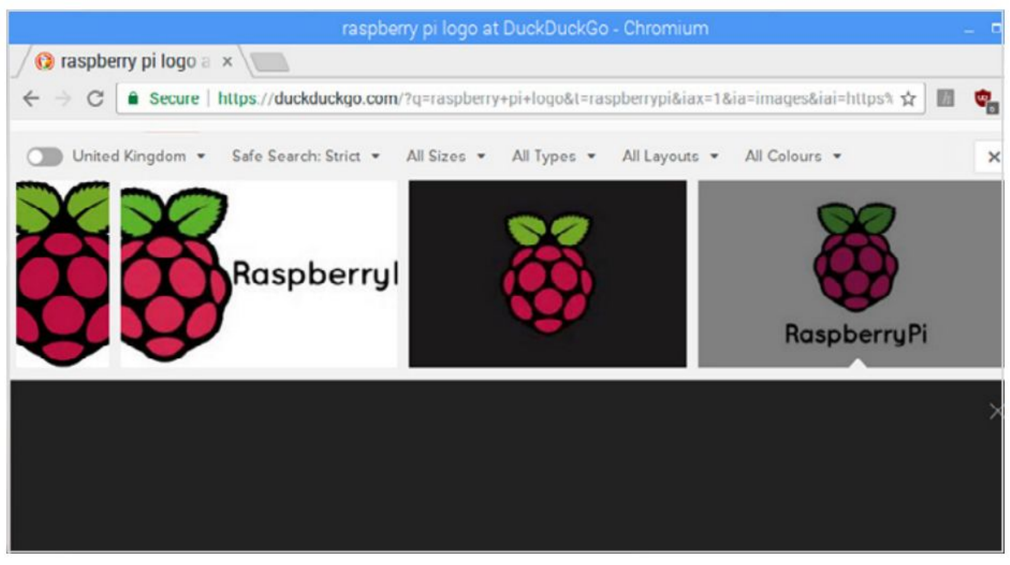
```
from turtle import *
color(‘red’, ‘yellow’)
begin_fill()
while True:
  forward(200)
  left(170)
  if abs(pos()) < 1:
    break
end_fill()
done()
```

It’s a different method, but very effective.



STEP 7

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



STEP 8

Now let’s get the code by importing the pygame module:

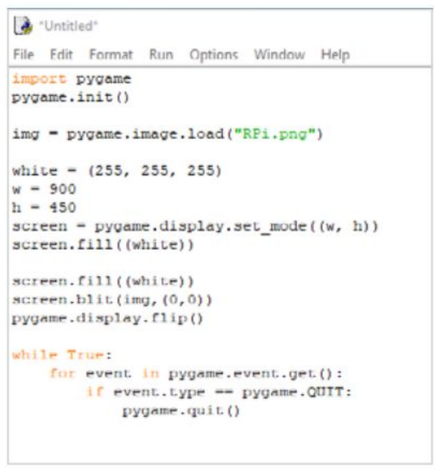
```
import pygame
pygame.init()
```

```
img = pygame.image.load(“RPi.png”)
```

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

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

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



STEP 9

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

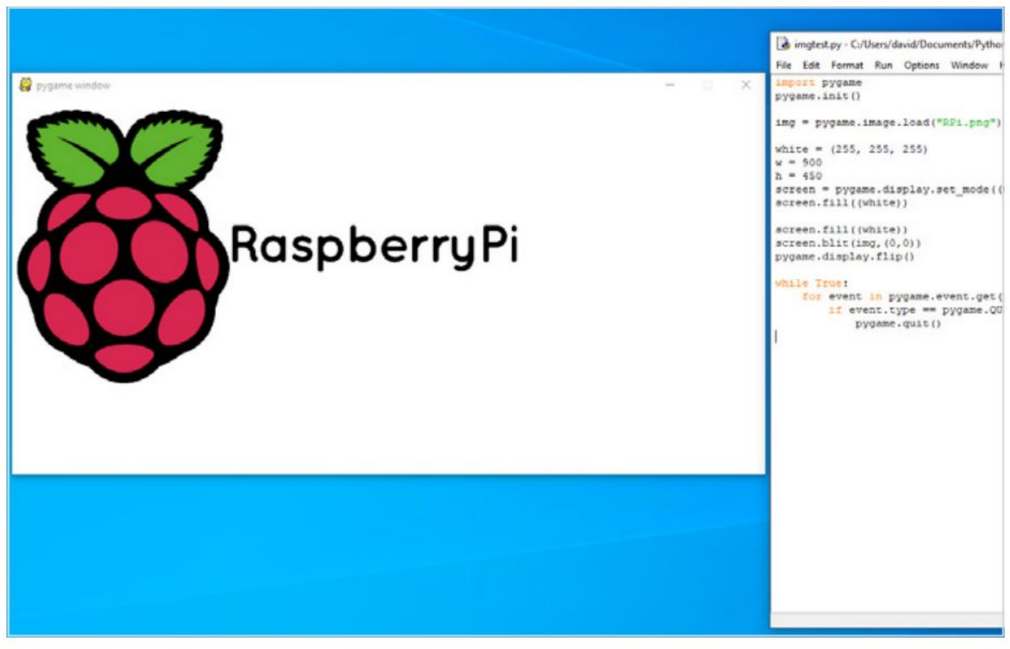
```
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

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

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

STEP 10

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



Combining What You Know So Far

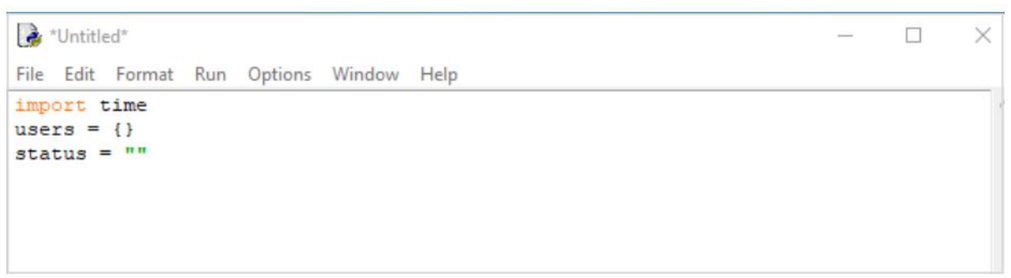
Based on what we've looked at over this section, let's combine it all and come up with a piece of code that can easily be applied into a real-world situation; or at the very least, something which you can incorporate into your programs.

LOGGING IN

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

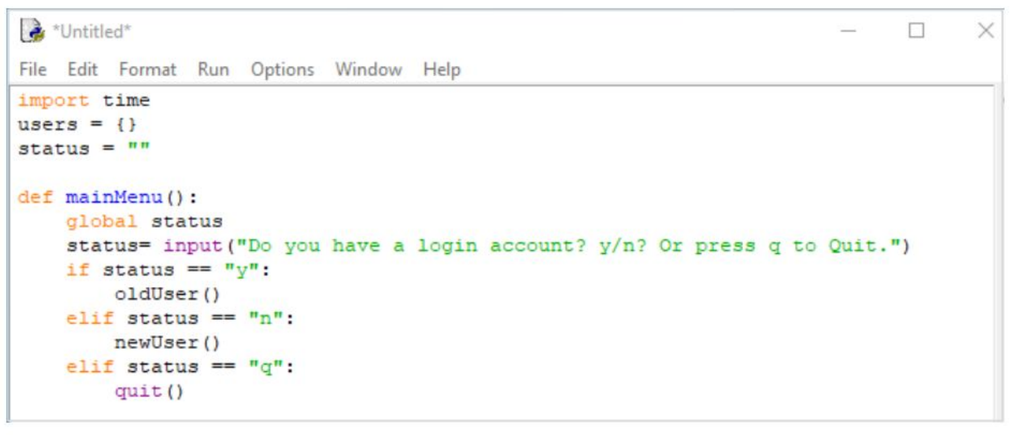
STEP 1 Let's begin by importing the Time module, creating a new dictionary to handle the usernames and passwords, and creating a variable to evaluate the current status of the program:

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



STEP 2 Next we need to define some functions. We can begin with creating the main menu, to where, after selecting the available options, all users will return:

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



STEP 3 The **global status** statement separates a local variable from one that can be called throughout the code, this way we can use the q=quit element without it being changed inside the function. We've also referenced some newly defined functions: oldUser and newUser which we'll get to next.

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

STEP 4 The newUser function is next:

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

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

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

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

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

STEP 5 Since we're using a Raspberry Pi, you will need to specify your own location for the logins.txt file. Essentially, this adds the username and password inputs, from the user, to the existing **users{}** dictionary. Therefore, the key and value structure remains; each user is the key, the password is the value.

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

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

STEP 6 Now to create the oldUser function:

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

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

```
    elif status == "n":
        newUser()
    elif status == "q":
        quit()

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

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

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

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

STEP 7 There's a fair bit happening here. We have login and passw variables, which are then matched to the users dictionary. If there's a match, then we have a successful login and the time and date of the login is outputted. If they don't match, then we print an error, and the process starts again.

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

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

STEP 8 Finally, we need to continually check that the 'q' key hasn't been pressed to exit the program. We can do this with:

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

```
*login.py - C:/Users/david/Documents/Python/login.py (3.7.0)*
File  Edit  Format  Run  Options  Window  Help
import time
users = {}
status = ""

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

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

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

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

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

STEP 9 Although a seemingly minor two lines, the **while** loop is what keeps the program running. At the end of every function, it's checked against the current value of status. If that global value isn't 'q' then the program continues. If it's equal to 'q' then the program can quit.

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

STEP 10 You can now create users, and then login with their names and passwords, with the logins.txt file being created to store the login data and successful logins being time-stamped. Now it's up to you to further improve the code. Perhaps you can import the list of created users from a previous session and, upon a successful login, display a graphic?