**LEARN PYTHON: FILES**

**Reading a File**

Computers use file systems to store and retrieve data. Each [file](https://www.codecademy.com/resources/docs/python/files?page_ref=catalog) is an individual container of related information. If you’ve ever saved a document, downloaded a song, or even sent an email you’ve created a file on some computer somewhere. Even **script.py**, the Python program you’re editing in the learning environment, is a file.

So, how do we interact with files using Python? We’re going to learn how to read and write different kinds of files using code. Let’s say we had a file called **real\_cool\_document.txt** with these contents:

**real\_cool\_document.txt**

Wowsers!

We could read that file like this:

**script.py**

with open('real\_cool\_document.txt') as cool\_doc:  
  cool\_contents = cool\_doc.read()  
print(cool\_contents)

This opens a file object called cool\_doc and creates a new indented block where you can read the contents of the opened file. We then read the contents of the file cool\_doc using cool\_doc.read() and save the resulting string into the variable cool\_contents. Then we print cool\_contents, which outputs the statement Wowsers!.

**Instructions**

**1.**

Use with to open the file **welcome.txt**. Save the file object as text\_file.

Checkpoint 2 Passed

Hint

Use this syntax to open a file:

with open('filename.txt') as file\_object:  
  # indented block here

**2.**

Read the contents of text\_file and save the results in text\_data.

Checkpoint 3 Passed

Hint

Use this syntax on a file object to read the contents of the file into a variable:

file\_string = file\_object.read()

**3.**

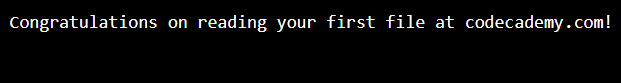
Print out text\_data.

**script.py**

with open('welcome.txt') as text\_file:

  text\_data = text\_file.read()

print(text\_data)

****

**Iterating Through Lines**

When we read a file, we might want to grab the whole document in a single string, like .read() would return. But what if we wanted to store each line in a variable? We can use the .readlines() function to read a text file line by line instead of having the whole thing. Suppose we have a file:

**keats\_sonnet.txt**

To one who has been long in city pent,  
’Tis very sweet to look into the fair  
And open face of heaven,—to breathe a prayer  
Full in the smile of the blue firmament.

**script.py**

with open('keats\_sonnet.txt') as keats\_sonnet:  
  for line in keats\_sonnet.readlines():  
    print(line)

The above script creates a temporary file object called keats\_sonnet that points to the file **keats\_sonnet.txt**. It then iterates over each line in the document and prints the entire file out.

**Instructions**

**1.**

Using a with statement, create a file object pointing to the file **how\_many\_lines.txt**. Store that file object in the variable lines\_doc.

Checkpoint 2 Passed

Hint

Remember to open a file using with syntax:

with open('filename.txt') as file\_object:  
  # indented block here

**2.**

Iterate through each of the lines in lines\_doc.readlines() using a for loop.

Inside the for loop print out each line of **how\_many\_lines.txt**.

Checkpoint 3 Passed

Hint

You can use the following syntax to print out each line of a file:

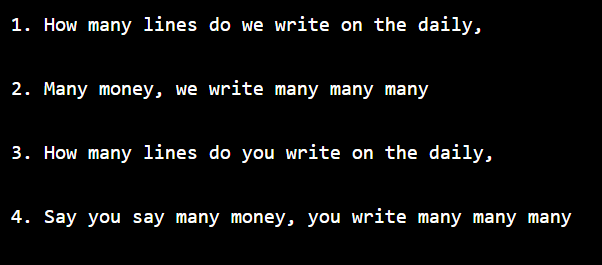
for line in file\_object.readlines():  
  print(line)

**script.py**

with open('how\_many\_lines.txt') as lines\_doc:

  for line in lines\_doc.readlines():

    print(line)

****

**Reading a Line**

Sometimes you don’t want to iterate through a whole file. For that, there’s a different file method, [.readline()](https://www.codecademy.com/resources/docs/python/files/readline?page_ref=catalog), which will only read a single line at a time. If the entire document is read line by line in this way subsequent calls to .readline() will not throw an error but will start returning an empty string (""). Suppose we had this file:

**millay\_sonnet.txt**

I shall forget you presently, my dear,  
So make the most of this, your little day,  
Your little month, your little half a year,  
Ere I forget, or die, or move away,

**script.py**

with open('millay\_sonnet.txt') as sonnet\_doc:  
  first\_line = sonnet\_doc.readline()  
  second\_line = sonnet\_doc.readline()  
  print(second\_line)

This script also creates a file object called sonnet\_doc that points to the file **millay\_sonnet.txt**. It then reads in the first line using sonnet\_doc.readline() and saves that to the variable first\_line. It then saves the second line (So make the most of this, your little day,) into the variable second\_line and then prints it out.

**Instructions**

**1.**

Using a with statement, create a file object pointing to the file **just\_the\_first.txt**. Store that file object in the variable first\_line\_doc.

Checkpoint 2 Passed

Hint

Remember to open a file using with syntax:

with open('filename.txt') as file\_object:  
  # indented block here

**2.**

Save the first line of **just\_the\_first.txt** into the variable first\_line.

Checkpoint 3 Passed

Hint

Use the file object method .readline() to store a line into the variable line.

**3.**

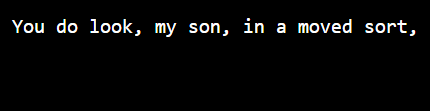
Print out the variable first\_line.

**script.py**

with open('just\_the\_first.txt') as first\_line\_doc:

  first\_line = first\_line\_doc.readline()

  print(first\_line)

****

**Writing a File**

Reading a file is all well and good, but what if we want to create a file of our own? With Python we can do just that. It turns out that our [open()](https://www.codecademy.com/resources/docs/python/built-in-functions/open?page_ref=catalog) function that we’re using to open a file to read needs another argument to open a file to write to.

**script.py**

with open('generated\_file.txt', 'w') as gen\_file:  
  gen\_file.write("What an incredible file!")

Here we pass the argument 'w' to open() in order to indicate to open the file in write-mode. The default argument is 'r' and passing 'r' to open() opens the file in read-mode as we’ve been doing.

This code creates a new file in the same folder as *script.py* and gives it the text What an incredible file!. It’s important to note that if there is already a file called *generated\_file.txt* it will completely overwrite that file, erasing whatever its contents were before.

**Instructions**

**1.**

Create a file object for the file **bad\_bands.txt** using the open() function with the w argument. Assign this object to the temporary variable bad\_bands\_doc.

Checkpoint 2 Passed

Hint

Remember to use the with statement to open a file and pass the 'w' argument to open it in write mode:

with open('file\_to\_write.txt', 'w') as file\_object:  
  # indent

**2.**

Use the bad\_bands\_doc.write() method to add the name of a musical group you dislike to the document bad\_bands.

**script.py**

with open('bad\_bands.txt', 'w') as bad\_bands\_doc:

  bad\_bands\_doc.write("Poison")

**Appending to a File**

So maybe completely deleting and overwriting existing files is something that bothers you. Isn’t there a way to just add a line to a file without completely deleting it? Of course there is! Instead of opening the file using the argument 'w' for write-mode, we open it with 'a' for append-mode. If we have a generated file with the following contents:

**generated\_file.txt**

This was a popular file...

Then we can add another line to that file with the following code:

**script.py**

with open('generated\_file.txt', 'a') as gen\_file:  
  gen\_file.write("\n... and it still is")

In the code above we open a file object in the temporary variable gen\_file. This variable points to the file *generated\_file.txt* and, since it’s open in append-mode, adds the string \n... and it still is to the file. The newline character \n moves to the next line before adding the rest of the string. If you were to open the file after running the script it would look like this:

**generated\_file.txt**

This was a popular file...  
... and it still is

Notice that opening the file in append-mode, with 'a' as an argument to open(), means that using the file object’s [.write()](https://www.codecademy.com/resources/docs/python/files/write?page_ref=catalog) method *appends* whatever is passed to the end of the file. If we were to run **script.py** again, this would be what **generated\_file.txt** looks like:

**generated\_file.txt**

This was a popular file...  
... and it still is  
... and it still is

Notice that we’ve appended "\n... and it still is" to the file a second time! This is because in **script.py** we opened **generated\_file.txt** in append-mode.

**Instructions**

**1.**

We’ve got a file, **cool\_dogs.txt**, filled with all the cool dogs we know. Somehow while compiling this list we forgot about one very cool dog. Let’s fix that problem by adding him to our **cool\_dogs.txt**.

Open up our file **cool\_dogs.txt** in append-mode and assign it to the file object cool\_dogs\_file.

Checkpoint 2 Passed

**2.**

Inside your with block, add “Air Buddy\n” to **cool\_dogs.txt**. Air Buddy is a Golden Retriever that plays basketball, which more than qualifies him for this list. The \n character moves to the next line after appending the string.

Checkpoint 3 Passed

Hint

Use cool\_dogs\_file.write() to add a string to the end of **cool\_dogs.txt**

**script.py**

with open('cool\_dogs.txt', 'a') as cool\_dogs\_file:

  cool\_dogs\_file.write("Air Buddy\n")

**What's With "with"?**

We’ve been opening these files with this with block so far, but it seems a little weird that we can only use our file variable in the indented block. Why is that? The with keyword invokes something called a *context manager* for the file that we’re calling open() on. This context manager takes care of opening the file when we call open() and then closing the file after we leave the indented block.

Why is closing the file so complicated? Well, most other aspects of our code deal with things that Python itself controls. All the variables you create: integers, lists, dictionaries — these are all Python objects, and Python knows how to clean them up when it’s done with them. Since your files exist *outside* your Python script, we need to tell Python when we’re done with them so that it can close the connection to that file. Leaving a file connection open unnecessarily can affect performance or impact other programs on your computer that might be trying to access that file.

The with syntax replaces older ways to access files where you need to call [.close()](https://www.codecademy.com/resources/docs/python/files/close?page_ref=catalog) on the file object manually. We can still open up a file and append to it with the old syntax, as long as we remember to close the file connection afterwards.

fun\_cities\_file = open('fun\_cities.txt', 'a')  
  
# We can now append a line to "fun\_cities".  
fun\_cities\_file.write("Montréal")  
  
# But we need to remember to close the file  
fun\_cities\_file.close()

In the above script we added “Montréal” as a new line in our file **fun\_cities.txt**. However, since we used the older-style syntax, we had to remember to close the file afterwards. Since this is necessarily more verbose (requires at least one more line of code) without being any more expressive, using with is preferred.

**Instructions**

**1.**

In **script.py** there’s a file object that doesn’t get closed correctly. Let’s fix it by changing the syntax!

Remove this line:

close\_this\_file = open('fun\_file.txt')

And change it to use the with syntax from our previous exercises.

Remember to indent the rest of the body so that we don’t get an IndentError.

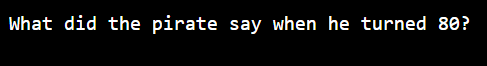
**script.py**

with open("fun\_file.txt") as close\_this\_file:

  setup = close\_this\_file.readline()

  punchline = close\_this\_file.readline()

print(setup)

****