EXERCISE 15

Reading Files

Everything you've learned about raw_input and argv is so you can start reading files. You may have to play with this exercise the most to understand what's going on, so do it carefully and remember your checks. Working with files is an easy way to *erase your work* if you are not careful.

This exercise involves writing two files. One is your usual ex15.py file that you will run, but the *other* is named ex15_sample.txt. This second file isn't a script but a plain text file we'll be reading in our script. Here are the contents of that file:

```
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.
```

What we want to do is "open" that file in our script and print it out. However, we do not want to just "hard code" the name ex15_sample.txt into our script. "Hard coding" means putting some bit of information that should come from the user as a string right in our program. That's bad because we want it to load other files later. The solution is to use argv and raw_input to ask the user what file the user wants instead of "hard coding" the file's name.

ex15.py

```
1     from sys import argv
2
3     script, filename = argv
4
5     txt = open(filename)
6
7     print "Here's your file %r:" % filename
8     print txt.read()
9
10    print "Type the filename again:"
11    file_again = raw_input("> ")
12
13    txt_again = open(file_again)
14
15    print txt_again.read()
```

A few fancy things are going on in this file, so let's break it down real quick:

Lines 1–3 should be a familiar use of argv to get a filename. Next we have line 5 where we use a new command open. Right now, run pydoc open and read the instructions. Notice how like your own scripts and raw_input, it takes a parameter and returns a value you can set to your own variable. You just opened a file.

Line 7 we print a little line, but on line 8 we have something very new and exciting. We call a function on `txt`. What you got back from open is a `file`, and it's also got commands you can give it. You give a file a command by using the `.` (dot or period), the name of the command, and parameters. Just like with open and `raw_input`. The difference is that when you say `txt.read()` you are saying, "Hey txt! Do your read command with no parameters!"

The remainder of the file is more of the same, but we'll leave the analysis to you in the Study Drills.

What You Should See

I made a file called "ex15_sample.txt" and ran my script.

Exercise 15 Session

```
$ python ex15.py ex15_sample.txt
Here's your file 'ex15_sample.txt':
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.


Type the filename again:
>  ex15_sample.txt
This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.
```

Study Drills

This is a big jump, so be sure you do this Study Drill as best you can before moving on.

1. Above each line, write out in English what that line does.
2. If you are not sure, ask someone for help or search online. Many times searching for "python THING" will find answers for what that THING does in Python. Try searching for "python open."
3. I used the name "commands" here, but they are also called "functions" and "methods." Search around online to see what other people do to define these. Do not worry if they confuse you. It's normal for programmers to confuse you with vast extensive knowledge.
4. Get rid of the part from lines 10–15 where you use `raw_input` and try the script then.
5. Use only `raw_input` and try the script that way. Think of why one way of getting the filename would be better than another.

6. Run pydoc `file` and scroll down until you see the `read()` command (method/function). See all the other ones you can use? Skip the ones that have __ (two underscores) in front because those are junk. Try some of the other commands.
7. Start `python` again and use open from the prompt. Notice how you can open files and run `read` on them right there?
8. Have your script also do a `close()` on the `txt` and `txt_again` variables. It's important to close files when you are done with them.

Common Student Questions

Does `txt = open(filename)` return the contents of the file?
No, it doesn't. It actually makes something called a "file object." You can think of it like an old tape drive that you saw on mainframe computers in the 1950s or even like a DVD player from today. You can move around inside them, and then "read" them, but the file is not the contents.

I can't type code into my Terminal/PowerShell like you say in Study Drill #7.
First thing, from the command line just type `python` and hit Enter. Now you are in `python` as we've done a few other times. Once you have that you can just type in code and Python will run it in little pieces. Play with that. To get out of it type `quit()` and hit Enter.

What does `from sys import argv` mean?
For now, just understand that `sys` is a package, and this phrase just says to get the `argv` feature from that package. You'll learn more about these later.

I put the name of the file in as `script, ex15_sample.txt = argv` but it doesn't work.
No, that's not how you do it. Make the code exactly like mine, then run it from the command line the exact same way I do. You don't put the names of files in; you let Python put the name in.

Why is there no error when we open the file twice?
Python will not restrict you from opening a file more than once, and in fact sometimes this is necessary.

This page intentionally left blank

EXERCISE 16

Reading and Writing Files

If you did the Study Drills from the last exercise, you should have seen all sorts of commands (methods/functions) you can give to files. Here's the list of commands I want you to remember:

- close—Closes the file. Like `File->Save..` in your editor.
- read—Reads the contents of the file. You can assign the result to a variable.
- readline—Reads just one line of a text file.
- truncate—Empties the file. Watch out if you care about the file.
- write(stuff)—Writes stuff to the file.

For now, these are the important commands you need to know. Some of them take parameters, but we do not really care about that. You only need to remember that `write` takes a parameter of a string you want to write to the file.

Let's use some of this to make a simple little text editor:

ex16.py

```python
from sys import argv

script, filename = argv

print "We're going to erase %r." % filename
print "If you don't want that, hit CTRL-C (^C)."
print "If you do want that, hit RETURN."

raw_input("?")

print "Opening the file..."
target = open(filename, 'w')

print "Truncating the file.  Goodbye!"
target.truncate()

print "Now I'm going to ask you for three lines."

line1 = raw_input("line 1: ")
line2 = raw_input("line 2: ")
line3 = raw_input("line 3: ")

print "I'm going to write these to the file."

target.write(line1)
target.write("\n")
```

```
27  target.write(line2)
28  target.write("\n")
29  target.write(line3)
30  target.write("\n")
31
32  print "And finally, we close it."
33  target.close()
```

That's a large file—probably the largest you have typed in. So go slow, do your checks, and make it run. One trick is to get bits of it running at a time. Get lines 1–8 running, then five more, then a few more, and so on, until it's all done and running.

What You Should See

There are actually two things you will see. First the output of your new script:

Exercise 16 Session

```
$ python ex16.py test.txt
We're going to erase 'test.txt'.
If you don't want that, hit CTRL-C (^C).
If you do want that, hit RETURN.
?
Opening the file...
Truncating the file.  Goodbye!
Now I'm going to ask you for three lines.
line 1:  Mary had a little lamb
line 2:  It's fleece was white as snow
line 3:  It was also tasty
I'm going to write these to the file.
And finally, we close it.
```

Now, open up the file you made (in my case `test.txt`) in your editor and check it out. Neat, right?

Study Drills

1. If you feel you do not understand this, go back through and use the comment trick to get it squared away in your mind. One simple English comment above each line will help you understand or at least let you know what you need to research more.
2. Write a script similar to the last exercise that uses `read` and `argv` to read the file you just created.
3. There's too much repetition in this file. Use strings, formats, and escapes to print out `line1`, `line2`, and `line3` with just one `target.write()` command instead of six.

4. Find out why we had to pass a `'w'` as an extra parameter to open. Hint: open tries to be safe by making you explicitly say you want to write a file.
5. If you open the file with `'w'` mode, then do you really need the `target.truncate()`? Go read the docs for Python's open function and see if that's true.

Common Student Questions

Is the `truncate()` necessary with the `'w'` parameter?
See Study Drills #5.

What does `'w'` mean?
It's really just a string with a character in it for the kind of mode for the file. If you use `'w'`, then you're saying "open this file in 'write' mode"—hence the `'w'` character. There's also `'r'` for "read," `'a'` for append, and modifiers on these.

What are the modifiers to the file modes we can use?
The most important one to know for now is the + modifier, so you can do `'w+'`, `'r+'`, and `'a+'`. This will open the file in both read and write mode and, depending on the character used, position the file in different ways.

Does just doing open`(filename)` open it in `'r'` (read) mode?
Yes, that's the default for the `open()` function.

This page intentionally left blank

EXERCISE 17

More Files

Now let's do a few more things with files. We're going to actually write a Python script to copy one file to another. It'll be very short but will give you some ideas about other things you can do with files.

ex17.py

```
from sys import argv
from os.path import exists

script, from_file, to_file = argv

print "Copying from %s to %s" % (from_file, to_file)

# we could do these two on one line too, how?
in_file = open(from_file)
indata = in_file.read()

print "The input file is %d bytes long" % len(indata)

print "Does the output file exist? %r" % exists(to_file)
print "Ready, hit RETURN to continue, CTRL-C to abort."
raw_input()

out_file = open(to_file, 'w')
out_file.write(indata)

print "Alright, all done."

out_file.close()
in_file.close()
```

You should immediately notice that we `import` another handy command named `exists`. This returns `True` if a file exists, based on its name in a string as an argument. It returns `False` if not. We'll be using this function in the second half of this book to do lots of things, but right now you should see how you can import it.

Using `import` is a way to get tons of free code other better (well, usually) programmers have written so you do not have to write it.

What You Should See

Just like your other scripts, run this one with two arguments: the file to copy from and the file to copy it to. I'm going to use a simple test file named `test.txt` again:

Exercise 17 Session

```
$ cat test.txt
This is a test file.
$
$ python ex17.py test.txt new_file.txt
Copying from test.txt to new_file.txt
The input file is 21 bytes long
Does the output file exist? False
Ready, hit RETURN to continue, CTRL-C to abort.

Alright, all done.
```

It should work with any file. Try a bunch more and see what happens. Just be careful you do not blast an important file.

WARNING! Did you see that trick I did with `cat` to show the file? You can learn how to do that in the appendix.

Study Drills

1. Go read up on Python's `import` statement, and start `python` to try it out. Try importing some things and see if you can get it right. It's alright if you do not.
2. This script is *really* annoying. There's no need to ask you before doing the copy, and it prints too much out to the screen. Try to make it more friendly to use by removing features.
3. See how short you can make the script. I could make this one line long.
4. Notice at the end of the WYSS I used something called `cat`? It's an old command that "con*cat*enates" files together, but mostly it's just an easy way to print a file to the screen. Type `man cat` to read about it.
5. Windows people, find the alternative to `cat` that Linux/OSX people have. Do not worry about `man` since there is nothing like that.
6. Find out why you had to do `output.close()` in the code.

Common Student Questions

Why is the `'w'` in quotes?
That's a string. You've been using them for a while now, so make sure you know what a string is.

No way you can make this one line!
That ; depends ; on ; how ; you ; define ; one ; line ; of ; code.

What does the `len()` function do?
It gets the length of the string that you pass to it and then returns that as a number. Play with it.

When I try to make this script shorter, I get an error when I close the files at the end.
You probably did something like this, `indata = open(from_file).read()`, which means you don't need to then do `in_file.close()` when you reach the end of the script. It should already be closed by Python once that one line runs.

Is it normal to feel like this exercise was really hard?
Yes, it is totally normal. Programming may not "click" for you until maybe even Exercise 36, or it might not until you finish the book and then make something with Python. Everyone is different, so just keep going and keep reviewing exercises that you had trouble with until it clicks. Be patient.

I get a `Syntax:EOL while scanning string literal` error.
You forgot to end a string properly with a quote. Go look at that line again.