Writing to Text Files

WE'LL COVER THE FOLLOWING

You can write to files in much the same way that you read from them. First you open a file and get a stream object, then you use methods on the stream object to write data to the file, then you close the file.

To open a file for writing, use the open() function and specify the write mode. There are two file modes for writing:

- Write" mode will overwrite the file. Pass `mode='w'` to the `open()` function.
- "Append" mode will add data to the end of the file. Pass `mode='a'` to the `open()` function.

Either mode will create the file automatically if it doesn't already exist, so there's never a need for any sort of fiddly "if the file doesn't exist yet, create a new empty file just so you can open it for the first time" function. Just open a file and start writing.

Just open a file and start writing.

You should always close a file as soon as you're done writing to it, to release the file handle and ensure that the data is actually written to disk. As with reading data from a file, you can call the stream object's `close()` method, or you can use the `with` statement and let Python close the file for you. I bet you can guess which technique I recommend.

```
with open('test.log', mode='w', encoding='utf-8') as a_file:  #①
```

```
    a_file.write('test succeeded')                          #②
with open('test.log', encoding='utf-8') as a_file:
    print(a_file.read())

#test succeeded

with open('test.log', mode='a', encoding='utf-8') as a_file:  #③
    a_file.write('and again')

with open('test.log', encoding='utf-8') as a_file:
    print(a_file.read())
#test succeededand again
```

① You start boldly by creating the new file `test.log` (or overwriting the existing file), and opening the file for writing. The `mode='w'` parameter means open the file for writing. Yes, that's all as dangerous as it sounds. I hope you didn't care about the previous contents of that file (if any), because that data is gone now.

② You can add data to the newly opened file with the `write()` method of the stream object returned by the `open()` function. After the `with` block ends, Python automatically closes the file.

③ That was so fun, let's do it again. But this time, with `mode='a'` to append to the file instead of overwriting it. Appending will never harm the existing contents of the file.

④ Both the original line you wrote and the second line you appended are now in the file `test.log`. Also note that neither carriage returns nor line feeds are included. Since you didn't write them explicitly to the file either time, the file doesn't include them. You can write a carriage return with the `'\r'` character, and/or a line feed with the `'\n'` character. Since you didn't do either, everything you wrote to the file ended up on one line.

Character Encoding Again

Did you notice the `encoding` parameter that got passed in to the `open()` function while you were opening a file for writing? It's important; don't ever leave it out! As you saw in the beginning of this chapter, files don't contain strings, they contain bytes. Reading a "string" from a text file only works because you told Python what encoding to use to read a stream of bytes and convert it to a string. Writing text to a file presents the same problem in

reverse. You can't write characters to a file; characters are an abstraction. In

order to write to the file, Python needs to know how to convert your string into a sequence of bytes. The only way to be sure it's performing the correct conversion is to specify the `encoding` parameter when you open the file for writing.