Lesson 1, Week 4: text file I/O

AIM

— To learn methods for opening, reading from and writing to text files

After this lesson, you will be able to

* Discuss why file I/O is fundamentally risky for a computer
* Describe the different kinds of access to files: read-only, write-only or read/write access, discarding or retaining file data
* Use `open` to create a file handle, specifying what access is required
* To use use `read` and `readlines` to read from a file with an open file handle
* To use `write` to write to a file with an open file handle
* To use `close` to close a file handle

File I/O in general

So far, you've only dealt with programs that generate their own data. But of course most of the time, working with data means importing data from files and exporting data to files—that is, file I/O.

You should think of a file arriving in your Julia program's workspace as a stranger arriving at your door: it could the package that you've been waiting for, it could be trouble. In the case of the stranger, you may spend a while talking to them before you open the door. For better or for worse, computers tend not to do that sort of thing[1]—instead, the designers of most programming languages, Julia included, put the responsibility for avoiding trouble on you, the programmer.

In practice, this means having a pretty good idea that the files you are about to use are not too large, that the files have no malicious content, and that you know the format of the file. On this course, we

[1] Except when establishing internet connections and the like.

only one format only: flat text files, which consist solely of characters[2], but many other formats are used—in particular, specialised formats limited to specific purposes and domains.

Files to be used for I/O arrive in your computer in many ways: Internet downloads, memory devices such as hard disks and USB sticks, measurement devices, cameras and more. However, once they're there the same thing applies to all of them: you open a connection to the file, your read and or write to the file, and you close the connection.

- The important point in opening the connection to the file is that you use the correct path from your working directory to the directory where the file is. The best is simply to put in the same working directory as your code.
- In Julia, there are many functions that read from and read to a file, once a connection to the file exists. We discuss a few of them below.
- Many of the problems with file I/O come from file connections that remain open unduly long. None of the ways to make sure this never happens are both easy and foolproof. On this course, we recommend that you open a file, read or write to it, and close it. This can be cumbersome and for some purposes completely unsuitable, but is close to foolproof.

Opening and closing files

Long experience in computing have led to the following situation: you as the programmer decide, at the time of opening the connection to a file, whether it is for reading only, or writing only, or both. Additionally, if it is for writing, you decide whether to overwrite the file or to add to it (inserting data in the middle of a file is not possible in Julia).

You open a connection as follows: `<fh> = open(<fname>, <read/write mode>)`.

Here `<fh>` is the name of the connection you make, `<fname>` is the name of the file in question, and `<read/write mode>` is a string that sets the type of access to the file.

The file handle `<fh>` is the name of the connection, and it has to follow the usual rules for naming a variable, and the read/write mode must be one of several options—let us use `?` to see them. Note that the connection is a variable, with a name and a value[3]

The `read` and `readlines` functions

We make available to you a text file containing the first two chapters of Pride and Prejudice by Jane Austen. The file is called `prideandprejudiceextract.txt`. It was created by simply cutting and pasting from the text file of the book at Project Gutenberg: `https://www.gutenberg.org/ebooks/1342`. There are many books available there, completely free—of course you should acknowledge both the website and the author if you use their material, as we have done here.

[2] It is true that some of them are escape sequences.

[3] Of type `IOStream`, if you must know.

We can read the whole text as a single string:

```
fh = open("prideandprejudiceextract.txt")
prideandprejudiceraw = read(fh, String)
close(fh)
```

The file handle `fh` is a name that refers to the *connection* to the file, not the file itself. Recall that a connection is a variable with a name and a value. It exists quite independently of the file.

Note also that there is no read/write code string in the call to `open` — this means that Julia uses the default code string, which is `"r"` and stands for "read-only". One cannot write to a file opened with a read-only file handle.

We can use the `split` function (more or less the opposite of `join`, see the `?` summary) to extract all the lines. This is such a common action that `readlines` is provided to get the same result with far less code. Lots of lines are empty, we use `filter!` to get rid of them. [DEMO]

Writing to a file

First, you need to open it with the correct read/write code string.

Julia uses the read/write code `"w"` to specify that you writing to a blank file. This is fine if you create a file with a new name, but if you happen to use the name of a file already there, the contents of that file will simply be ignored[4] and the file will contain only the data written after you connected with it.

In other words, when you specify `"w"`, you specify that existing data should be destroyed.

To add to an existing file, use the `"a"` read/write code string. [DEMO: the `?` guidance on]

Once you have an appropriate file handle, use the `write` function:

```
f = open("newfile.txt", "w")
write(f, "what a boring test string")
close(f)
```

DEMO: try several writes, try opening and closing several times, try with `"a"` read-/write code string; use `\n` to get new lines.

Review and summary

* File I/O has three stages: opening a connection to a file, reading and/or writing to the file, closing the connection
* Files that stay open when they should be closed lead to computer problems
* The programmer decides, at the time of opening the connection, whether the access is for reading, for writing, or for both
* If the access is for writing, the programmer must also decide whether to keep existing data in the file or not

[4] The file content doesn't immediately disappear, however. This is why recovery of accidentally erased files is sometimes possible.

* The syntax is `<file handle> open(<file name>, <read/write code>)`
* If you omit the read/write code string, the default `"r"` is used, which means "read-only"
* Read the file as a single string[5] with `read(<file handle>, String)`
* Read the file as an array of strings with `readlines(<file handle>, String)`
* To write to file from scratch (destroying any data it may have contained), open it with `"w"`
* To append to a file (retaining all the data it contains), open it with `"a"`
* Use the syntax `write(<file handle>, <string value>)` to write a plain text file
* Appending happens on the same line as the last line in a file, unless that line ends with a newline character

[5] This assumes it is a text file, of course.