File I/O

Leen-Kiat Soh

Computer Science & Engineering

University of Nebraska, Lincoln, NE

Input/Output (I/O)

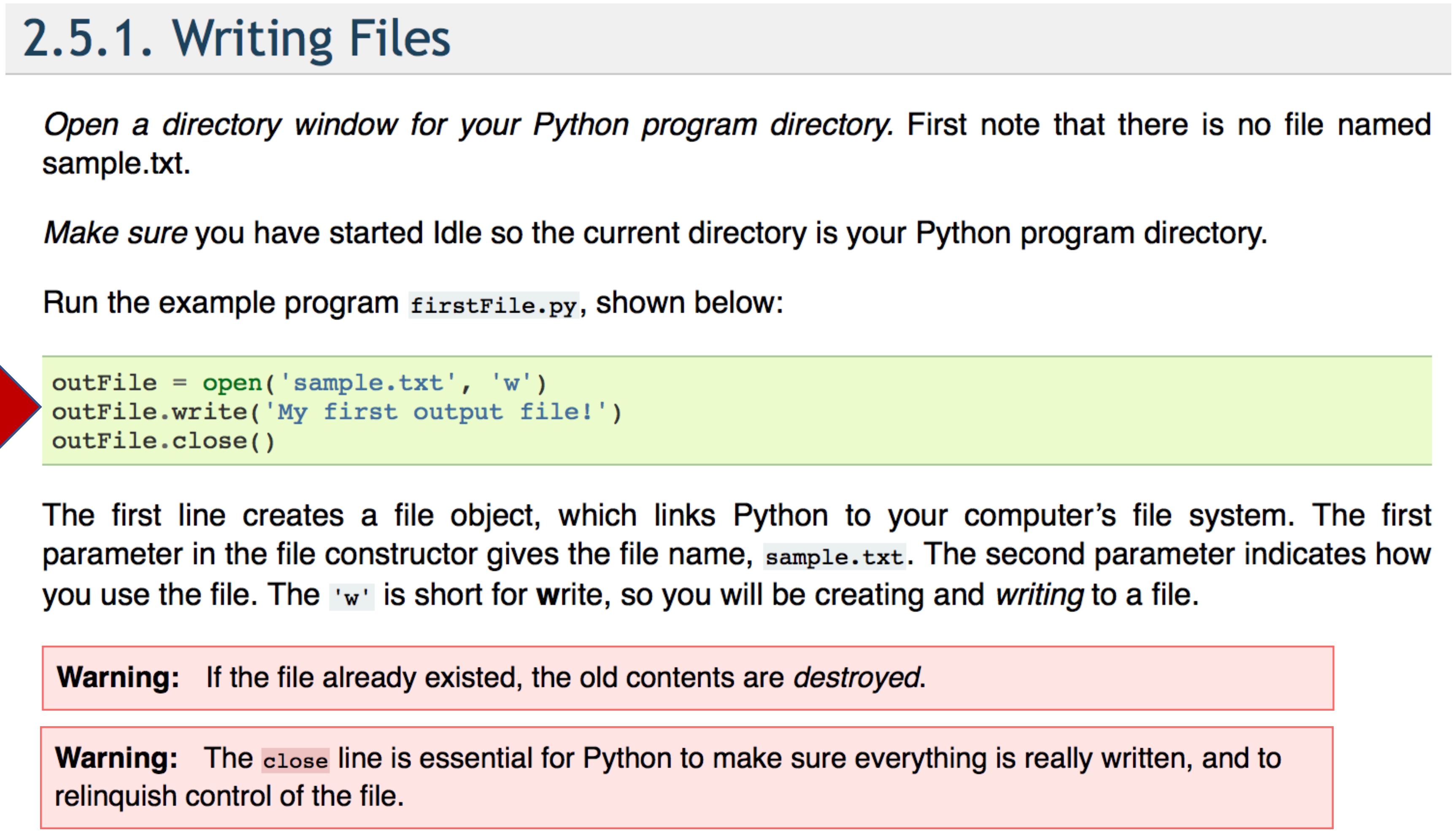
- Standard I/O
  - Printing/displaying *output* to the screen
  - Prompting user to enter response via keyboard typing
  - Processing the *input* entry accordingly
- File I/O
  - Reading in file(s) as input
  - Writing out results or derived data into a file as output
  - Reading in input data stream from other devices
  - Generating output data stream into other devices

Why File I/O?

- Data are typically stored as files
- Files allow for transfer and sharing of data more conveniently
- Reduce data entry errors
- Improve data processing throughput
- Improve data validation process
- Improve testing

Writing to an output file

Syntax



2.5.1. Writing Files

Open a directory window for your Python program directory. First note that there is no file named sample.txt.

Make sure you have started Idle so the current directory is your Python program directory.

Run the example program `firstFile.py`, shown below:

Syntax

```
outFile = open('sample.txt', 'w')
outFile.write('My first output file!')
outFile.close()
```

The first line creates a file object, which links Python to your computer's file system. The first parameter in the file constructor gives the file name, `sample.txt`. The second parameter indicates how you use the file. The `'w'` is short for **w**rite, so you will be creating and *writing* to a file.

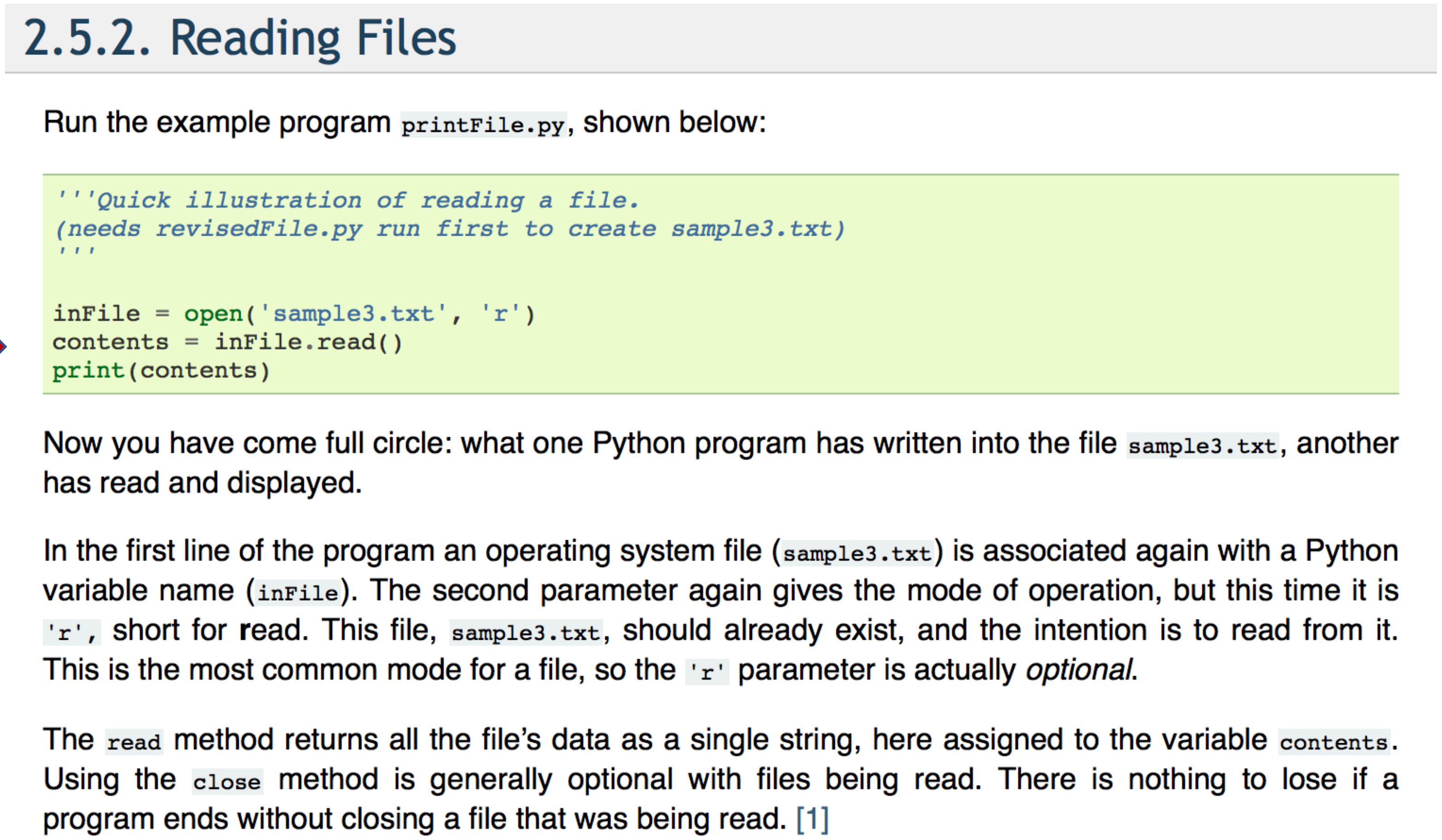
Warning: If the file already existed, the old contents are *destroyed.*

Warning: The `close` line is essential for Python to make sure everything is really written, and to relinquish control of the file.

http://anh.cs.luc.edu/handsonPythonTutorial/files.html

Reading from an input file Syntax

Syntax



2.5.2. Reading Files

Run the example program `printFile.py`, shown below:

```
'''Quick illustration of reading a file.
(needs revisedFile.py run first to create sample3.txt)
'''

inFile = open('sample3.txt', 'r')
contents = inFile.read()
print(contents)
```

Now you have come full circle: what one Python program has written into the file `sample3.txt`, another has read and displayed.

In the first line of the program an operating system file (`sample3.txt`) is associated again with a Python variable name (`inFile`). The second parameter again gives the mode of operation, but this time it is `'r'`, short for read. This file, `sample3.txt`, should already exist, and the intention is to read from it. This is the most common mode for a file, so the `'r'` parameter is actually *optional*.

The `read` method returns all the file's data as a single string, here assigned to the variable `contents`. Using the `close` method is generally optional with files being read. There is nothing to lose if a program ends without closing a file that was being read. [1]

http://anh.cs.luc.edu/handsonPythonTutorial/files.html

Important

- The **file name** is a string that identifies the file in your computer's file system
- You need the file name to open a file creating a **file** object, but the file object is ***not*** the same as the name of the file on your hard drive
  - **You assign it to a variable name just for use inside your program**
- The **contents** of the file
  - The read method for a file object reads and *returns* existing content, while the write method writes new content into the file

http://anh.cs.luc.edu/handsonPythonTutorial/loops.html