Chapter Seven

FILES AND EXCEPTIONS

Chapter Goals

- To read and write text files
- To process collections of data
- To process command line arguments
- To raise and handle exceptions

In this chapter, you will learn how to write programs that manipulate files

Contents

Reading and Writing Text Files

SECTION 7.1

Reading and Writing Text Files

- Text files are very commonly used to store information
  - They are the most ‘portable’ types of data files
- Examples of text files include files that are created with a simple text editor, such as Windows Notepad, and Python source code and HTML files

Opening Files: Reading

- To access a file, you must first *open* it
- Suppose you want to read data from a file named input.txt, located in the same directory as the program
- To open a file for reading, you must provide the name of the file as the first argument to the open function and the string "r" as the second argument:

```
infile = open("input.txt", "r")
```

Opening Files: Reading (2)

- Important things to keep in mind:
  - When opening a file for reading, the file must exist (and otherwise be accessible) or an exception occurs
  - The file object returned by the open function must be saved in a variable
    - All operations for accessing a file are made via the file object

Opening Files: Writing

- To open a file for writing, you provide the name of the file as the first argument to the open function and the string "w" as the second argument:

```
outfile = open("output.txt", "w")
```

- If the output file already exists, it is emptied before the new data is written into it
- If the file does not exist, an empty file is created

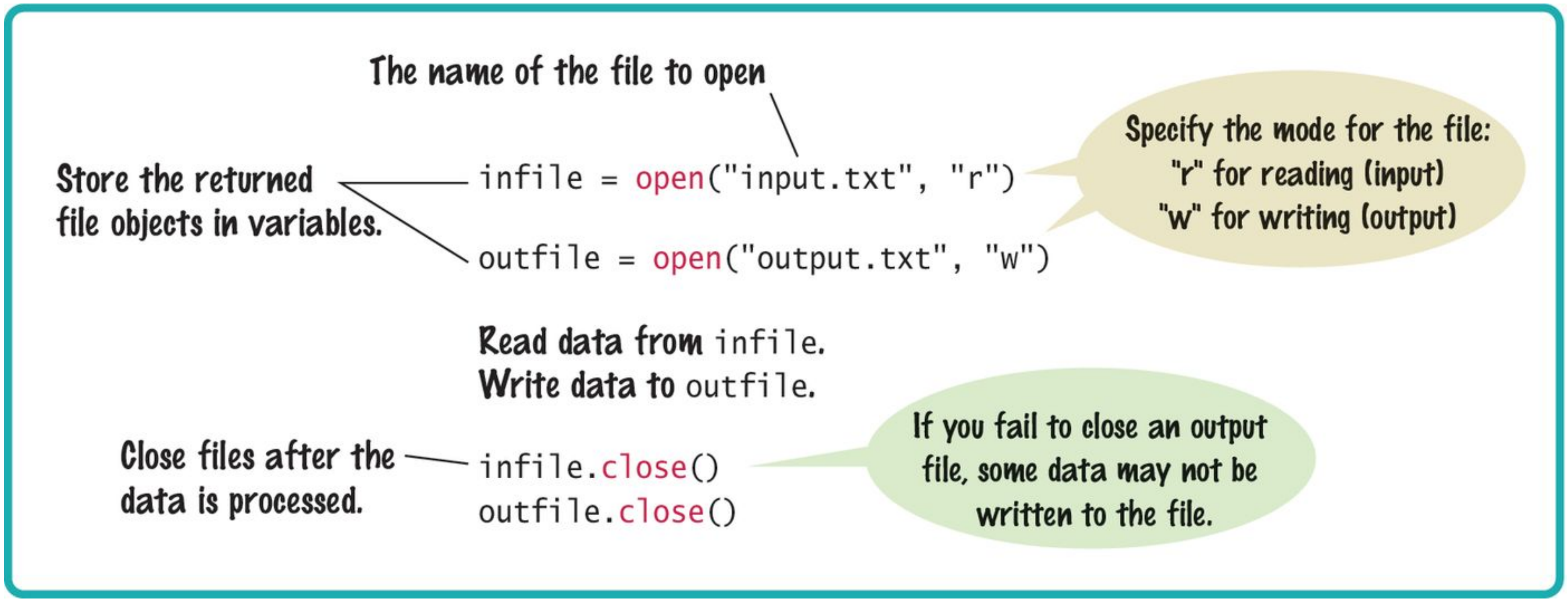
Closing Files: Important

- When you are done processing a file, be sure to *close* the file using the close() method:

```
infile.close()
outfile.close()
```

- If your program exits without closing a file that was opened for writing, some of the output may not be written to the disk file

Syntax: Opening And Closing Files


The name of the file to open
Store the returned file objects in variables.
Specify the mode for the file: "r" for reading (input) "w" for writing (output)
Close files after the data is processed.
If you fail to close an output file, some data may not be written to the file.

```
infile = open("input.txt", "r")
outfile = open("output.txt", "w")
```

Read data from `infile`.
Write data to `outfile`.

```
infile.close()
outfile.close()
```

Reading From a File

- To read a line of text from a file, call the readline() method with the file object that was returned when you opened the file:

```
line = infile.readline()
```

- When a file is opened, an input marker is positioned at the beginning of the file
- The readline() method reads the text, starting at the current position and continuing until the end of the line is encountered
  - The input marker is then moved to the next line

Reading From a File (2)

- For example, suppose input.txt contains the lines

  flying

  circus

- The first call to readline() returns the string "flying\n"
  - Recall that \n denotes the newline character that indicates the end of the line
- If you call readline() a second time, it returns the string "circus\n"

Reading From a File (3)

- Calling readline() again yields the empty string "" because you have reached the end of the file
- If the file contains a blank line, then readline() returns a string containing only the newline character "\n"

Reading Multiple Lines From a File

- You repeatedly read a line of text and process it until the sentinel value is reached:
- The sentinel value is an empty string, which is returned by the readline() method after the end of file has been reached

```
line = infile.readline()
while line != "" :
    # Process the line.
    line = infile.readline()
```