File and Exception Exercises

7

Files allow us to work with data, without needing to enter it each time our program runs. Files also allow us to store results from our program in a more permanent manner. These features are often used when creating larger programs. When completing the exercises in this chapter, you should expect to:

- Open a file for reading and/or writing
- Read data from a file
- Write data to a new file
- Use values provided to the program as command line parameters
- Detect and recover from errors such as attempting to open a file that doesn't exist
- Detect and recover from other errors that are not specifically related to files

Some of the exercises in this chapter involve reading from existing files such as a list of words, names or chemical elements. You can download these files from the author's website: http://www.cpsc.ucalgary.ca/~bdstephe/PythonWorkbook.

Exercise 141: Display the Head of a File

(*Solved—40 Lines*)

Unix-based operating systems usually include a tool named `head`. It displays the first 10 lines of a file whose name is provided as a command line parameter. Write a Python program that provides the same behavior. Display an appropriate error message if the file requested by the user does not exist or if the command line parameter is omitted.

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_7

Exercise 142: Display the Tail of a File

(*Solved—35 Lines*)

Unix-based operating systems also typically include a tool named `tail`. It displays the last 10 lines of a file whose name is provided as a command line parameter. Write a Python program that provides the same behavior. Display an appropriate error message if the file requested by the user does not exist or if the command line parameter is omitted.

There are several different approaches that can be taken to solve this problem. One option is to load the entire contents of the file into a list and then display the last 10 elements. Another option is to read the contents of the file twice, once to count the lines, and a second time to display the last 10 lines. However, both of these solutions are undesirable when working with large files. Another solution exists that only requires you to read the file once, and only requires you to store 10 lines from the file at one time. For an added challenge, develop such a solution.

Exercise 143: Concatenate Multiple Files

(*Solved—27 Lines*)

Unix-based operating systems typically include a tool named `cat`, which is short for concatenate. Its purpose is to concatenate and display one or more files whose names are provided as command line parameters. The files are displayed in the same order that they appear on the command line.

Create a Python program that performs this task. It should generate an appropriate error message for any file that cannot be displayed, and then proceed to the next file. Display an appropriate error message if your program is started without any command line parameters.

Exercise 144: Number the Lines in a File

(*23 Lines*)

Create a program that adds line numbers to a file. The name of the input file will be read from the user, as will the name of the new file that your program will create. Each line in the output file should begin with the line number, followed by a colon and a space, followed by the line from the input file.

Exercise 145: Find the Longest Word in a File

(*39 Lines*)

In this exercise you will create a Python program that identifies the longest word(s) in a file. Your program should output an appropriate message that includes the length of the longest word, along with all of the words of that length that occurred in the file. Treat any group of non-white space characters as a word, even if it includes numbers or punctuation marks.

Exercise 146: Letter Frequencies

(*43 Lines*)

One technique that can be used to help break some simple forms of encryption is frequency analysis. This analysis examines the encrypted text to determine which characters are most common. Then it tries to map the most common letters in English, such as `E` and `T`, to the most commonly occurring characters in the encrypted text.

Write a program that initiates this process by determining and displaying the frequencies of all letters in a file. Ignore spaces, punctuation marks, and numbers as you perform this analysis. Your program should be case insensitive, treating `a` and `A` as equivalent. The user will provide the file name as a command line parameter. Your program should display a meaningful error message if the user provides the wrong number of command line parameters, or if the program is unable to open the file indicated by the user.

Exercise 147: Words that Occur Most

(*37 Lines*)

Write a program that displays the word (or words) that occur most frequently in a file. Your program should begin by reading the name of the file from the user. Then it should find the word(s) by splitting each line in the file at each space. Finally, any leading or trailing punctuation marks should be removed from each word. In addition, your program should ignore capitalization. As a result, `apple`, `apple!`, `Apple` and `ApPlE` should all be treated as the same word. You will probably find your solution to Exercise 111 helpful when completing this problem.

Exercise 148: Sum a List of Numbers

(*Solved—26 Lines*)

Create a program that sums all of the numbers entered by the user while ignoring any lines entered by the user that are not valid numbers. Your program should display the current sum after each number is entered. It should display an appropriate error message after any invalid input, and then continue to sum any additional numbers entered by the user. Your program should exit when the user enters a blank line. Ensure that your program works correctly for both integers and floating point numbers.

Hint: This exercise requires you to use exceptions without using files.

Exercise 149: Both Letter Grades and Grade Points

(*106 Lines*)

Write a program that converts from letter grades to grade points and vice-versa. Your program will convert multiple values entered by the user, with one value entered on each line. Begin by attempting to convert each value entered by the user from a number of grade points to a letter grade. If an exception occurs during the attempt then your program should attempt to convert the value from a letter grade to a number of grade points. If both conversions fail then your program should provide a message indicating that the supplied input is invalid. Design your program so that it continues performing conversions until the user enters a blank line. Your solutions to Exercises 51 and 52 may be helpful when completing this exercise.

Exercise 150: Remove Comments

(*Solved—46 Lines*)

Python uses the # character to mark the beginning of a comment. The comment ends at the end of the line containing the # character. In this exercise, you will create a program that removes all of the comments from a Python source file. Check each line in the file to determine if a # character is present. If it is then your program should remove all of the characters from the # character to the end of the line (we'll ignore the situation where the comment character occurs inside of a string). Save the modified file using a new name that will be entered by the user. The user will also enter the name of the input file. Ensure that an appropriate error message is displayed if a problem is encountered while accessing the files.

Exercise 151: Two Word Random Password

(*Solved—37 Lines*)

While generating a password by selecting random characters generally gives a relatively secure password, it also generally gives a password that is difficult to memorize. As an alternative, some systems construct a password by taking two English words and concatenating them. While this password isn't as secure, it is much easier to memorize.

Write a program that reads a file containing a list of words, randomly selects two of them, and concatenates them to produce a new password. When producing the password ensure that the total length is between 8 and 10 characters, and that each word used is at least three letters long. Capitalize each word in the password so that the user can easily see where one word ends and the next one begins. Display the password for the user.

Exercise 152: What's that Element Again?

(*59 Lines*)

Write a program that reads a file containing information about chemical elements and stores it in one or more appropriate data structures. Then your program should read and process input from the user. If the user enters an integer then your program should display the symbol and name of the element with the number of protons entered. If the user enters a string then your program should display the number of protons for the element with that name or symbol. Your program should display an appropriate error message if no element exists for the name, symbol or number of protons entered. Continue to read input from the user until a blank line is entered.

Exercise 153: A Book with No "e" ...

(*Solved—49 Lines*)

The novel "Gadsby" is over 50,000 words in length. While 50,000 words isn't normally remarkable for a novel, it is in this case because none of the words in the book use the letter "e". This is particularly noteworthy when one considers that "e" is the most common letter in English.

Write a program that reads a list of words from a file and determines what proportion of the words use each letter of the alphabet. Display the result for all 26 letters. Include an additional message identifying the letter that is used in the smallest proportion of the words. Your program should ignore any punctuation marks and it should treat uppercase and lowercase letters as equivalent.

Exercise 154: Names that Reached Number One

(*Solved—50 Lines*)

The baby names data set consists of over 200 files. Each file contains a list of 100 names, along with the number of times each name was used. There are two files for each year: one containing names used for girls and the other containing names used for boys. The data set includes data for every year from 1900 to 2012.

Write a program that reads every file in the data set and identifies all of the names that were most popular in at least one year. Your program should output two lists: one containing the most popular names for boys and the other containing the most popular names for girls. Neither of your lists should include any repeated values.

Exercise 155: Gender Neutral Names

(*56 Lines*)

Some names, like Ben and Jonathan, are normally only used for boys while names like Rebbecca and Flora are normally only used for girls. Other names, like Chris and Alex, may be used for both boys and girls.

Write a program that determines and displays all of the baby names that were used for both boys and girls in a year specified by the user. Your program should generate an appropriate message if there were no gender neutral names in the selected year. Display an appropriate error message if you do not have data for the year requested by the user. Additional details about the baby names data set are included in Exercise 154.

Exercise 156: Most Births in a given Time Period

(*76 Lines*)

Write a program that uses the baby names data set described in Exercise 154 to determine which names were used most often within a time period. Have the user supply the first and last years of the range to analyze. Display the boy's name and the girl's name given to the most children during the indicated years.

Exercise 157: Distinct Names

(*41 Lines*)

In this exercise, you will create a program that reads every file in the baby names data set described in Exercise 154. As your program reads the files, it should keep track of each name used for a boy and each name used for a girl. Your program should

output two lists. One list will contain all of the names that have been used for girls. The other list will contain all of the names that have been used for boys. Neither of your lists should contain any repeated values.

Exercise 158: Spell Checker

(*Solved—51 Lines*)

A spell checker can be a helpful tool for people who struggle to spell words correctly. In this exercise, you will write a program that reads a file and displays all of the words in it that are misspelled. Misspelled words will be identified by checking each word in the file against a list of known words. Any words in the user's file that do not appear in the list of known words will be reported as spelling mistakes.

The user will provide the name of the file to check for spelling mistakes as a command line parameter. Your program should display an appropriate error message if the command line parameter is missing. An error message should also be displayed if your program is unable to open the user's file. Use your solution to Exercise 111 when creating your solution to this exercise so that words followed by a comma, period or other punctuation mark are not reported as spelling mistakes. Ignore the capitalization of the words when checking their spelling.

Hint: While you could load all of the English words from the words data set into a list, searching a list is slow if you use Python's `in` operator. It is much faster to check if a key is present in a dictionary, or if a value is present in a set. If you use a dictionary, the words will be the keys. The values can be the integer 0 (or any other value) because the values will never be used.

Exercise 159: Repeated Words

(*61 Lines*)

Spelling mistakes are only one of many different kinds of errors that might appear in a written work. Another error that is common for some writers is a repeated word. For example, an author might inadvertently duplicate a word, as shown in the following sentence:

```
At least one value must be entered
entered in order to compute the average.
```

Some word processors will detect this error and identify it when a spelling or grammar check is performed.

In this exercise you will write a program that detects repeated words in a text file. When a repeated word is found your program should display a message that contains the line number and the repeated word. Ensure that your program correctly handles the case where the same word appears at the end of one line and the beginning of the following line, as shown in the previous example. The name of the file to examine will be provided as the program's only command line parameter. Display an appropriate error message if the user fails to provide a command line parameter, or if an error occurs while processing the file.

Exercise 160: Redacting Text in a File

(*Solved—49 Lines*)

Sensitive information is often removed, or redacted, from documents before they are released to the public. When the documents are released it is common for the redacted text to be replaced with black bars.

In this exercise you will write a program that redacts all occurrences of sensitive words in a text file by replacing them with asterisks. Your program should redact sensitive words wherever they occur, even if they occur in the middle of another word. The list of sensitive words will be provided in a separate text file. Save the redacted version of the original text in a new file. The names of the original text file, sensitive words file, and redacted file will all be provided by the user.

> You may find the `replace` method for strings helpful when completing this exercise. Information about the `replace` method can either be found in your textbook or on the internet.

For an added challenge, extend your program so that it redacts words in a case insensitive manner. For example, if `exam` appears in the list of sensitive words then redact `exam`, `Exam`, `ExaM` and `EXAM`, among other possible capitalizations.

Exercise 161: Missing Comments

(*Solved—44 Lines*)

When one writes a function, it is generally a good idea to include a comment that outlines the function's purpose, its parameters and its return value. However, sometimes comments are forgotten, or left out by well-intentioned programmers that plan to write them later but then never get around to it.

Create a python program that reads one or more Python source files and identifies functions that are not immediately preceded by a comment. For the purposes of this exercise, assume that any line that begins with `def`, followed by a space, is the

beginning of a function definition. Assume that the comment character, #, will be the first character on the previous line when the function has a comment. Display the names of all of the functions that are missing comments, along with the file name and line number where the function definition is located.

The user will provide the names of one or more Python files as command line parameters. If your program encounters a file that doesn't exist or can't be opened then it should display an appropriate error message before moving on and processing the remaining files.

Exercise 162: Consistent Line Lengths

(*45 Lines*)

While 80 characters is a common width for a terminal window, some terminals are narrow or wider. This can present challenges when displaying documents containing paragraphs of text. The lines might be too long and wrap, making them difficult to read, or they might be too short and fail to make use of the available space.

Write a program that opens a file and displays it so that each line is filled as full as possible. If you read a line that is too long then your program should break it up into words and add words to the current line until it is full. Then your program should start a new line and display the remaining words. Similarly, if you read a line that is too short then you will need to use words from the next line of the file to finish filling the current line of output. For example, consider a file containing the following lines from "Alice's Adventures in Wonderland":

```
Alice was
beginning to get very tired of sitting by her
sister
on the bank, and of having nothing to do: once
or twice she had peeped into the book her sister
was reading, but it had
no
pictures or conversations in it,"and what is
the use of a book," thought Alice, "without
pictures or conversations?"
```

When formatted for a line length of 50 characters, it should be displayed as:

```
Alice was beginning to get very tired of sitting
by her sister on the bank, and of having nothing
to do: once or twice she had peeped into the book
her sister was reading, but it had no pictures or
conversations in it, "and what is the use of a
book," thought Alice, "without pictures or
conversations?"
```

Ensure that your program works correctly for files containing multiple paragraphs of text. You can detect the end of one paragraph and the beginning of the next by looking for lines that are empty once the end of line marker has been removed. You may perform error checking if you want to, but it is not required for this exercise.

Hint: Use a constant to represent the maximum line length. This will make it easier to update the program when the window size changes.

Exercise 163: Words with Six Vowels in Order

(*56 Lines*)

There is at least one word in the English language that contains each of the vowels *a*, *e*, *i*, *o*, *u* and *y* exactly once and in order. Write a program that searches a file containing a list of words and displays all of the words that meet this constraint. The user will provide the name of the file that will be searched. Display an appropriate error message and exit the program if the user provides an invalid file name or if something else goes wrong while searching for words with six vowels in order.