File and Exception Solutions

15

Solution to Exercise 141: Display the Head of a File

```
##
# Display the head (first 10 lines) of a file whose name is provided as a command line parameter.
#
import sys

NUM_LINES = 10

# Verify that exactly one command line parameter (in addition to the .py file) was supplied
if len(sys.argv) != 2:
  print("You must provide the file name as a command line parameter.")
  quit()

try:
  # Open the file for reading
  inf = open(sys.argv[1], "r")

  # Read the first line from the file
  line = inf.readline()

  # Keep looping until we have either read and displayed 10 lines or
  # we have reached the end of the file
  count = 0
  while count < NUM_LINES and line != "":
    # Remove the trailing newline character and count the line
    line = line.rstrip()
    count = count + 1

    # Display the line
    print(line)

    # Read the next line from the file
    line = inf.readline()

  # Close the file
  inf.close()
```

When the `quit` function is called the program ends immediately.

B. Stephenson, *The Python Workbook*, DOI 10.1007/978-3-319-14240-1_15

```
except IOError:
  # Display a message if something goes wrong while accessing the file
  print("An error occurred while accessing the file.")
```

Solution to Exercise 142: Display the Tail of a File

```
##
# Display the tail (last lines) of a file whose name is provided as a command line parameter.
#
import sys

NUM_LINES = 10

# Verify that exactly one command line parameter (in addition to the .py file) was provided
if len(sys.argv) != 2:
  print("The file name must be provided as a command line parameter.")
  quit()

try:
  # Open the file for reading
  inf = open(sys.argv[1], "r")

  # Read through the file, always saving the NUM_LINES most recent lines
  lines = []
  for line in inf:
    # Add the most recent line to the end of the list
    lines.append(line)
    # If we now have more than NUM_LINES lines then remove the oldest one
    if len(lines) > NUM_LINES:
      lines.pop(0)

  # Close the file
  inf.close()

except:
  print("An error occurred while processing the file.")
  quit()

# Display the last lines of the file
for line in lines:
  print(line, end="")
```

Solution to Exercise 143: Concatenate Multiple Files

```
##
# Concatenate one or more files, displaying the result.
#
import sys
```

```
# Ensure that at least one command line parameter (in addition to the .py file) has been provided
if len(sys.argv) == 1:
  print("You must provide at least one file name.")
  quit()

# Process all of the files provided on the command line
for i in range(1, len(sys.argv)):
  fname = sys.argv[i]
  try:
    # Open the current file for reading
    inf = open(fname, "r")

    # Display the file
    for line in inf:
      print(line, end="")

    # Close the file
    inf.close()

  except:
    # Display a message, but do not quit so that the program will go on to process subsequent files
    print("Couldn't open/display", fname)
```

The element at position 0 in `sys.argv` is the Python file that is being executed. As a result, our for loop starts processing files at position 1 in the list.

Solution to Exercise 148: Sum a List of Numbers

```
##
#  Compute the sum of numbers entered by the user, ignoring non-numeric input.
#

# Read the first line of input from the user
line = input("Enter a number: ")
total = 0

# Keep reading until the user enters a blank line
while line != "":
  try:
    # Try and convert the line to a number
    num = float(line)
    # If the conversion succeeds then add it to the total and display it
    total = total + num
    print("The total is now", total)

  except ValueError:
    # Display an error message before going on to read the next value
    print("That wasn't a number.")

  # Read the next number
  line = input("Enter a number: ")

# Display the total
print("The grand total is", total)
```

Solution to Exercise 150: Remove Comments

```
##
#  Remove all of the comments from a python file, ignoring the case where
#  a comment character occurs within a string.
#

# Read and open the input file, ensuring that it was opened successfully
try:
  in_name = input("Enter the name of a Python file: ")
  inf = open(in_name, "r")
except:
  print("A problem was encountered while opening the input file.")
  print("Quitting...")
  quit()

# Read and open the output file, ensuring that it was opened successfully
try:
  out_name = input("Enter the output file name: ")
  outf = open(out_name, "w")
except:
  print("A problem was encountered while opening the output file.")
  print("Quitting...")
  quit()

try:
  # Read all of the lines from the input file, process them to remove
  # comments, and save the lines to a new file
  for line in inf:
    # Find the position of the comment character (-1 if there isn't one)
    pos = line.find("#")

    # If there is a comment then form a slice of the
    # string that excludes it, overwriting line
    if pos > -1:
      line = line[0 : pos]
      line = line + "\n"

    # Write the (potentially modified) line to the file
    outf.write(line)

  # Close the files
  inf.close()
  outf.close()

except:
  print("A problem was encountered while processing the file.")
  print("Quitting...")
```

The position of the comment character is stored in `pos`. As a result, `line[0 : pos]` is all of the characters up to but not including the comment character.

Solution to Exercise 151: Two Word Random Password

```
##
#  Generate a password by concatenating two random words.  The password will be
#  between 8 and 10 characters, and each word will be at least three letters long.
#
from random import randrange

WORD_FILE = "../Data/words.txt"

# Read all of the words from the file, only keeping those that are
# between 3 and 7 characters in length, and store them in a list.
words = []
inf = open(WORD_FILE, "r")
for line in inf:
  # Remove the newline character
  line = line.rstrip()

  # Keep words that are between 3 and 7 letters long
  if len(line) >= 3 and len(line) <= 7:
    words.append(line)

# Close the file
inf.close()

# Randomly select the first word for the password.  It can be any word.
first = words[randrange(0, len(words))]
first = first.capitalize()

# Keep selecting a second word until we find one that doesn't make the
# password too short or too long
password = first
while len(password) < 8 or len(password) > 10:
  second = words[randrange(0, len(words))]
  second = second.capitalize()
  password = first + second

# Display the generated password
print("The random password is:", password)
```

The password we are creating will be between 8 and 10 characters in length. Since the shortest acceptable word is 3 letters long, and a password must have 2 words in it, a password can never contain a word longer than 7 letters.

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

```
##
#  Determine the proportion of words that include each letter of the alphabet.
#  The letter that is used in the smallest proportion of words is highlighted.
#
WORD_FILE = "../Data/words.txt"

# Maintain a dictionary that counts the number of words containing each letter.
# Initialize the count for each letter to 0.
contains = {}
for ch in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
  contains[ch] = 0
```

```
# Open the file, and process each word, updating the contains dictionary
num_words = 0
inf = open(WORD_FILE, "r")
for word in inf:
  # Convert the word to uppercase and remove the newline character
  word = word.upper().rstrip()

  # Before we can update the dictionary we need to generate a list of the unique letters in the
  # word.  Otherwise we will increase the count multiple times for words that contain repeated
  # letters.  We also need to remove any hyphens that might be present.
  unique = []
  for ch in word:
    if ch not in unique and ch != "-":
      unique.append(ch)

  # Now increment the counts for all of the letters that are in the list of unique characters
  for ch in unique:
    contains[ch] = contains[ch] + 1

  # Keep count of the number of words that we have processed
  num_words = num_words + 1

# Close the file
inf.close()

# Display the result for each letter.  While displaying the results we will also
# determine which character had the smallest count so that we can display it later.
smallest_count = min(contains.values())
for ch in sorted(contains):
  if contains[ch] == smallest_count:
    smallest_letter = ch
  percentage = contains[ch] / num_words * 100
  print(ch, "occurs in %.2f percent of words" % percentage)

# Display the letter that is easiest to avoid based on the number of words in which it appears.
print()
print("The letter that is easiest to avoid is", smallest_letter)
```

Solution to Exercise 154: Names that Reached Number One

```
##
#  Display all of the girls' and boys' names that were the most popular in at least one year between
# 1900 and 2012.
#
FIRST_YEAR = 1900
LAST_YEAR = 2012

##
#  Load the first line from the file, extract the name, and add it to the
#  names list if it is not already present.
#  @param the name of the file to read the data from
#  @param the list to add the name to (if it isn't already present)
#  @return (None)
def LoadAndAdd(fname, names):
  # Open the file, read the first line, and extract the name
  inf = open(fname, "r")
  line = inf.readline()
  inf.close()
  parts = line.split()
  name = parts[0]

  # Add the name to the list if it is not already present
  if name not in names:
    names.append(name)

# Display the girls and boys names that reached #1 in at least one year between 1900 and 2012
def main():
  # Create two lists to store the most popular names
  girls = []
  boys = []

  # Process each year in the range, reading the first line out of the girl file and the boy file
  for year in range(FIRST_YEAR, LAST_YEAR + 1):
    girl_fname = "../Data/BabyNames/" + str(year) + "_GirlsNames.txt"
    boy_fname = "../Data/BabyNames/" + str(year) + "_BoysNames.txt"
```

My solution assumes that the baby names data files are stored in a different folder than the Python program. If you have the data files in the same folder as your program then `../Data/BabyNames/` should be omitted.

```
    LoadAndAdd(girl_fname, girls)
    LoadAndAdd(boy_fname, boys)

  # Display the lists
  print("Girls' names that reached #1:")
  for name in girls:
    print(" ", name)
  print()
```

```
  print("Boys' names that reached #1: ")
  for name in boys:
    print(" ", name)

# Call the main function
main()
```

Solution to Exercise 158: Spell Checker

```
##
#  Find and list all of the words in a file that are misspelled.
#
from only_words import onlyTheWords
from sys import argv


WORDS_FILE = "../Data/words.txt"

# Ensure that the program has the correct number of command line parameters
if len(argv) != 2:
  print("One command line parameter must be provided.  Quiting...")
  quit()

# Open the file.  Quit if the file is not opened successfully.
try:
  inf = open(argv[1], "r")
except:
  print("Failed to open '%s' for reading.  Quiting..." % argv[1])
  quit()

# Load all of the words into a dictionary of valid words.  The
# value 0 is associated with each word, but it is never used.
valid = {}
words_file = open(WORDS_FILE, "r")
for word in words_file:
  word = word.lower().rstrip()
  valid[word] = 0
words_file.close()

# Read each line from the file, adding any misspelled
# words to the list of misspellings
misspelled = []
for line in inf:
  # Discard the punctuation marks by calling the function developed in Exercise 111
  words = onlyTheWords(line)
  for word in words:
    # Only add to the misspelled list if the word is misspelled and not already in the list
    if word.lower() not in valid and word not in misspelled:
      misspelled.append(word)

# Close the file being checked
inf.close()
```

This solution uses a dictionary, but the values in the dictionary are never used. As a result, a set would be a better choice if you have learned about that data structure. A list is not used because checking if a key is in a dictionary is faster than checking if an element is in a list.

```
# Display the misspelled words, or a message indicating that no words are misspelled
if len(misspelled) == 0:
  print("No words were misspelled.")
else:
  print("The following words are misspelled:")
  for word in misspelled:
    print(" ", word)
```

Solution to Exercise 160: Redacting Text in a File

```
##
# Redact a text file by removing all occurrences of sensitive words.
# The redacted version of the text is written to a new file.
#
# Note that this program does not perform any error checking, and it
# does not implement case insensitive redaction.
#

# Get the name of the input file and open it
inf_name = input("Enter the name of the text file to redact: ")
inf = open(inf_name, "r")

# Get the name of the sensitive words file and open it
sen_name = input("Enter the name of the sensitive words file: ")
sen = open(sen_name, "r")

# Load all of the sensitive words into a list
words = []
line = sen.readline()
while line != "":
  line = line.rstrip()
  words.append(line)

  line = sen.readline()

sen.close()

# Get the name of the output file and open it
outf_name = input("Enter the name for the new redacted file: ")
outf = open(outf_name, "w")

# Read each line from the input file.  Replace all of the sensitive words
# with asterisks.  Then write the line to the output file.
line = inf.readline()
while line != "":
  # Check for and replace each sensitive word.  Use a number of asterisks that
  # matches the number of letters in the word
  for word in words:
    line = line.replace(word, "*" * len(word))
```

The sensitive word file can be closed now because all of the words have been read into a list.

```
  # Write the modified line to the output file
  outf.write(line)

  # Read the next line from the input file
  line = inf.readline()

# Close the input and output files
inf.close()
outf.close()
```

Solution to Exercise 161: Missing Comments

```
##
#  Find and display the names of Python functions that are not immediately preceded by a comment.
#
from sys import argv

# Verify that at least one file name has been provided as a command line parameter
if len(argv) == 1:
  print("At least one filename must be provided as a", \
        "command line parameter.")
  print("Quiting...")
  quit()

# Process each file provided as a command line parameter
for fname in argv[1 : len(argv)]:
  # Attempt to process the file
  try:
    inf = open(fname, "r")

    # As we move through the file we need to keep a copy of the previous
    # line so that we can check to see if it starts with a comment character.
    # We also need to keep track of the line number within the file.
    prev = " "
    lnum = 1
```

The `prev` variable must be initialized to a string that is at least one character in length. Otherwise the program will crash when the first line in the file that is being checked is a function definition.

```
    # Check each line in the current file
    for line in inf:
      # If the line is a function definition and the previous line is not a comment
      if line.startswith("def ") and prev[0] != "#":
        # Find the first ( on the line so that we know where the function name ends
        bracket_pos = line.index("(")
        name = line[4 : bracket_pos]

        # Display information about the function that is missing its comment
        print("%s line %d: %s" % (fname, lnum, name))
```

```
      # Save the current line and update the line counter
      prev = line
      lnum = lnum + 1

    # Close the current file
    inf.close()

except:
    print("A problem was encountered with file '%s'." % fname)
    print("Moving on to the next file...")
```