Chapter 7
Files and Exceptions

STARTING OUT WITH

Python

First Edition

by
Tony Gaddis

7.1 Introduction to File Input and Output

Concept:

When a program needs to save data for later use, it writes the data in a file. The data can be read from the file at a later time.

1-2

7-2

7.1 Introduction to File Input and Output

Terms

- **Saving data in a file = "writing data to" the file**
- ***Output file* = file that data is written to**
- **Retrieving data from a file = "reading data from" the file**
- ***Input file* = file that data is read from**

1-3

7-3

7.1 Introduction to File Input and Output

Types of Files

Two types of files:

- ***Text file* - contains data that has been encoded as text, ASCII or Unicode**
- ***Binary file* - contains data that has not been converted to text**

1-4

7-4

7.1 Introduction to File Input and Output

File Access Methods

Two ways to access data stored in files:

•*Sequential Access* - access data from the beginning of the file to the end of the file

•*Direct (random) Access*- directly access any piece of data in the file without reading the data that comes before it

1-5

7-5

7.1 Introduction to File Input and Output

Opening a File

The `open` function:

•General format:
```
file_variable = open(filename, mode)
```

•`file_variable` is the name of the variable that will reference the file object

•`filename` is a string specifying the name of the file

•`mode` is a string specifying the mode (reading, writing, etc.)

1-6

7-6

7.1 Introduction to File Input and Output

Table 7-1 Some of the Python file modes

Mode	Description
`'r'`	Open a file for reading only. The file cannot be changed or written to.
`'w'`	Open a file for writing. If the file already exists, erase its contents. If it does not exist, create it.
`'a'`	Open a file to be written to. All data written to the file will be appended to its end. If the file does not exist, create it.

customer_file* = `open`*(‘customers.txt’, ‘r’)

sales_file* = `open`*(‘sales.txt’, ‘w’)

1-7

7.1 Introduction to File Input and Output

Writing Data to a File

Method **- function that belongs to an object**

- perform operation using that object

`file_variable.write(string)`

- ***`file_variable`*** **– variable that references a file object**
- ***`write`*** **- file object used to write data to a file**
- ***`string`*** **- string that will be written to the file**

1-8

7.1 Introduction to File Input and Output

Program 7-1 (file_write.py)

```
# This program writes three lines of data
# to a file.
def main():
    # Open a file named philosophers.txt.
    outfile = open('philosophers.txt', 'w')

    # Write the names of three philosphers
    # to the file.
    outfile.write('John Locke\n')
    outfile.write('David Hume\n')
    outfile.write('Edmund Burke\n')

    # Close the file.
    outfile.close()

# Call the main function.
main()
```

1-9

7.1 Introduction to File Input and Output

Reading Data from a File … `read method`

Program 7-2 (file_read.py)

```
# This program reads and displays the contents
# of the philosophers.txt file.
def main():
    # Open a file named philosophers.txt.
    infile = open('philosophers.txt', 'r')

    # Read the file's contents.
    file_contents = infile.read()

    # Close the file.
    infile.close()

    # Print the data that was read into
    # memory.
    print file_contents

# Call the main function.
main()
```

1-10

7.1 Introduction to File Input and Output

Reading Data from a File … `readline method`

Program 7-3 (line_read.py)

```
# This program reads the contents of the
# philosophers.txt file one line at a time.
def main():
    # Open a file named philosophers.txt.
    infile = open('philosophers.txt', 'r')

    # Read three lines from the file.
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()

    # Close the file.
    infile.close()

    # Print the data that was read into
    # memory.
    print line1
    print line2
    print line3

# Call the main function.
main()
```

1-11

7.1 Introduction to File Input and Output

Concatenating a Newline to a String

```
myfile.write(name1 + ‘\n’)
```

Reading a String and Stripping the Newline From It

```
line1 = infile.readline()

# Strip the \n from the string
line1 = line1.rstrip(‘\n’)
```

1-12

7.1 Introduction to File Input and Output

Appending Data to an Existing File

•'a' mode to open an output file in append mode

•If the file already exists, it will not be erased

•If the file does not exist, it will be created

•When data is written to the file, it will be written at the end of the file's current contents.

7.1 Introduction to File Input and Output

Writing and Reading Numeric Data

- Numbers must be converted to strings before they can be written
- Built-in function, `str`, converts a value to a string

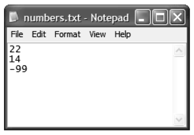
```
outfile.write(str(num1) + '\n')
```

7.1 Introduction to File Input and Output

Writing and Reading Numeric Data

```
infile = open('number.txt', 'r')

# readline method reads strings
string_input = infile.readline()

# built-in function int converts string to an integer
value = int(string_input)
```

Figure 7-15 The `numbers.txt` file viewed in Notepad



```
22
14
-99
```

7.1 Introduction to File Input and Output

Writing and Reading Numeric Data

Program 7-7
(read_numbers.py)

```
# This program demonstrates how numbers that are
# read from a file must be converted from strings
# before they are used in a math operation.

def main():
    # Open a file for reading.
    infile = open('numbers.txt', 'r')

    # Read three numbers from the file.
    num1 = int(infile.readline())
    num2 = int(infile.readline())
    num3 = int(infile.readline())

    # Close the file.
    infile.close()

    # Add the three numbers.
    total = num1 + num2 + num3

    # Display the numbers and their total.
    print 'The numbers are:', num1, num2, num3
    print 'Their total is:', total

# Call the main function.
main()
```

7.2 Using Loops to Process Files

Concept:

Files usually hold large amounts of data, and programs typically use a loop to process the data in a file.

7.2 Using Loops to Process Files

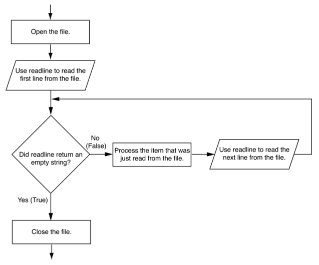
Reading a File with a Loop and Detecting the End of the File

- Read the contents of a file without knowing the number of items that are stored in the file
- `readline` method returns an empty string (' ') when it attempts to read beyond the end of file
- Priming read is required to test loop condition, if using a `while` loop

7.2 Using Loops to Process Files

Reading a File with a Loop and Detecting the End of the File

Figure 7-17 General logic for detecting the end of a file


Open the file.
Use readline to read the first line from the file.
Did readline return an empty string?
No (False)
Process the item that was just read from the file.
Use readline to read the next line from the file.
Yes (True)
Close the file.

7.2 Using Loops to Process Files

Using Python's `for` Loop to Read Lines

•`for` loop automatically reads a line of text from the input file

•No special condition or testing is needed

•No priming read is needed

•Automatically stops when the end of file is reached

```
for variable in file_object:
     statement
     statement
     etc.
```

7.2 Using Loops to Process Files

Using Python's `for` Loop to Read Lines

Program 7-10
(read_sales2.py)

```
# This program uses the for loop to read
# all of the values in the sales.txt file.

def main():
    # Open the sales.txt file for reading.
    sales_file = open('sales.txt', 'r')

    # Read all the lines from the file.
    for line in sales_file:
        # Convert line to a float.
        amount = float(line)
        # Format and display the amount.
        print '$%.2f' % amount

    # Close the file.
    sales_file.close()

# Call the main function.
main()
```

7.3 Processing Records

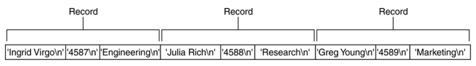
Concept:

The data that is stored in a file is frequently organized in records. A record is a complete set of data about an item, and a field is an individual piece of data within a record.

7.3 Processing Records

- A file's data is organized into records and fields
- ***Record*** – a complete set of data that describes one item
- ***Field*** – a single piece of data within a record

Figure 7-19 Records in a file



7.3 Processing Records

File manipulations

- **Create** an employee records file
- **Read** an employee records file
- **Add** records to a file
- **Search** for a record in a file
- **Modify** a record in a file
- **Delete** a record in a file

7.3 Processing Records

Creating an employee records file … **employee.txt**

1. **Get the total number of employees**
2. **Open the file for writing**
3. **For each employee**
   a. **Get the data for an employee**
   b. **Pad each field with newline character**
   c. **Write the employee record to the file**
4. **Close the file**

7.3 Processing Records

Creating an employee records file … **employee.txt**

Program 7-13
(save_emp_records.py)

```python
# This program gets employee data from the user and
# saves it as records in the employee.txt file.

def main():
    # Get the number of employee records to create.
    num_emps = input('How many employee records ' + \
                     'do you want to create? ')

    # Open a file for writing.
    emp_file = open('employees.txt', 'w')

    # Get each employee's data and write it to
    # the file.
    for count in range(1, num_emps + 1):
    # Get the data for an employee.
    print 'Enter data for employee #' + str(count)
    name = raw_input('Name: ')
    id_num = raw_input('ID number: ')
    dept = raw_input('Department: ')

    # Write the data as a record to the file.
    emp_file.write(name + '\n')
    emp_file.write(id_num + '\n')
    emp_file.write(dept + '\n')

    # Display a blank line.
    print

    # Close the file.
    emp_file.close()
    print 'Employee records written to employees.txt.'

# Call the main function.
main()
```

7.3 Processing Records

Reading an employee records file … **employee.txt**

1. **Open the file for reading**
2. **Read the first line from file**
3. **While NOT end-of-file**
   a. **Read employee record**
   b. **Strip the newlines from the each field**
   c. **Display employee record**
4. **Close the file**

7.3 Processing Records

Reading an employee records file … **employee.txt**

Program 7-14
(read_emp_records.py)

```python
# This program displays the records that are
# in the employees.txt file.

def main():
    # Open the employees.txt file.
    emp_file = open('employees.txt', 'r')

    # Read the first line from the file, which is
    # the name field of the first record.
    name = emp_file.readline()

    # If a field was read, continue processing.
    while name != '':
        # Read the ID number field.
        id_num = emp_file.readline()

        # Read the department field.
        dept = emp_file.readline()

        # Strip the newlines from the fields.
        name = name.rstrip('\n')
        id_num = id_num.rstrip('\n')
        dept = dept.rstrip('\n')

        # Display the record.
        print 'Name:', name
        print 'ID:', id_num
        print 'Dept:', dept
        print

        # Read the name field of the next record.
        name = emp_file.readline()

    # Close the file.
    emp_file.close()

# Call the main function.
main()
```

7.3 Processing Records

Add records to a file … **coffee.txt**

1. **Set flag to Yes**
2. **Open the file for appending**
3. **While flag is Yes**
   a. **Get coffee record**
   b. **Append coffee record**
   c. **Determine whether user wants to add another record**
4. **Close the file**

7.3 Processing Records

Add records to a file … **coffee.txt**

Program 7-15
(add_coffee_record.py)

```python
# This program adds coffee inventory records to
# the coffee.txt file.

def main():
    # Create a variable to control the loop.
    another = 'y'

    # Open the coffee.txt file in append mode.
    coffee_file = open('coffee.txt', 'a')

    # Add records to the file.
    while another == 'y' or another == 'Y':
        # Get the coffee record data.
        print 'Enter the following coffee data:'
        descr = raw_input('Description: ')
        qty = input('Quantity (in pounds): ')

        # Append the data to the file.
        coffee_file.write(descr + '\n')
        coffee_file.write(str(qty) + '\n')

        # Determine whether the user wants to add
        # another record to the file.
        print 'Do you want to add another record?'
        another = raw_input('Y = yes, anything else = no: ')

    # Close the file.
    coffee_file.close()
    print 'Data appended to coffee.txt.'

# Call the main function.
main()
```

7.3 Processing Records

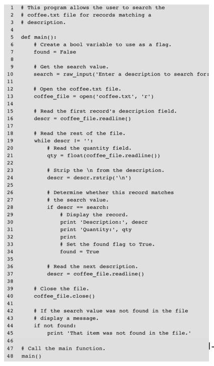
Searching for a record in a file … **coffee.txt**

1. **Set a flag to False**
2. **Get the search value**
3. **Open the file for reading**
4. **Read the first line from file**
5. **While NOT end-of-file**
   a. **Read coffee record**
   b. **Strip the newlines from the each field**
   c. **Determine whether the record is a match to search value**
      i. **If match - Set flag to True**
   d. **Read the next line from the file**
6. **Close the file**
7. **If flag is False – Display message "Not Found"**

7.3 Processing Records

Searching for a record in a file … **coffee.txt**

Program 7-17
(search_coffee_records.py)



```
# This program allows the user to search the
# coffee.txt file for records matching a
# description.

def main():
    # Create a bool variable to use as a flag.
    found = False

    # Get the search value.
    search = raw_input('Enter a description to search for:

    # Open the coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Read the first record's description field.
    descr = coffee_file.readline()

    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

        # Determine whether this record matches
        # the search value.
        if descr == search:
            # Display the record.
            print 'Description:', descr
            print 'Quantity:', qty
            print
            # Set the found flag to True.
            found = True

        # Read the next description.
        descr = coffee_file.readline()

    # Close the file.
    coffee_file.close()

    # If the search value was not found in the file
    # display a message.
    if not found:
        print 'That item was not found in the file.'

# Call the main function.
main()
```

7.3 Processing Records

Modify a record in a file … **coffee.txt**

1. **Set a flag to False**
2. **Get the search value**
3. **Open the original file for reading**
4. **Open a temporary file for writing**
5. **Read the first line from file**
6. **While NOT end-of-file**
   a. **Read coffee record**
   b. **Strip the newlines from the each field**
   c. **Determine whether the record is a match to search value**
      i. **If match – Write modified record to temporary file; Set flag to True**
      ii. **If NOT match – Write current record to temporary file**
   d. **Read the next line from the file**
7. **Close the original file**
8. **Close the temporary file**
9. **Delete original file**
10. **Rename temporary to the original file name**
11. **If flag is True – Display message "File updated"**
12. **If flag is False – Display message "Not Found"**

7.3 Processing Records

Modify a record in a file … **coffee.txt**

- **Requires a temporary file**
- **Copy all record from existing file to temporary file**
  - **BUT do not copy the contents of the modified record**
  - **Write the new data for the modified record**
- **Delete the original file …**
  - **Import Python's `os` module**
  - **Use the `remove` function**
    `os.remove`*`(original_file_name)`*
- **Rename the temporary file to the original file name …**
  - **Import Python's `os` module**
  - **Use the `rename` function**
    `os.rename`*`(temporary_file_name, original_file_name)`*

7.3 Processing Records

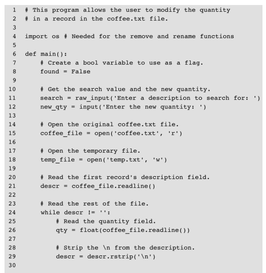
Modify a record in a file … **coffee.txt**

1. **Import `os` module**
2. **Set a flag to False**
3. **Get the search value**
4. **Open the original file for reading**
5. **Open a temporary file for writing**
6. **Read the first line from file**
7. **While NOT end-of-file**
   a. **Read coffee record**
   b. **Strip the newlines from the each field**
   c. **Determine whether the record is a match to search value**
      i. **If match – Write modified record to temporary file; Set flag to True**
      ii. **If NOT match – Write current record to temporary file**
   d. **Read the next line from the file**
8. **Close the original file**
9. **Close the temporary file**
10. **Delete original file**
11. **Rename temporary to the original file name**
12. **If flag is True – Display message "File updated"**
13. **If flag is False – Display message "Not Found"**

7.3 Processing Records

Modify a record in a file … **coffee.txt**

Program 7-18
(modify_coffee_records.py)

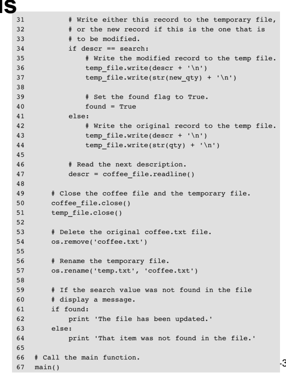


```
# This program allows the user to modify the quantity
# in a record in the coffee.txt file.

import os # Needed for the remove and rename functions

def main():
    # Create a bool variable to use as a flag.
    found = False

    # Get the search value and the new quantity.
    search = raw_input('Enter a description to search for: ')
    new_qty = input('Enter the new quantity: ')

    # Open the original coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Open the temporary file.
    temp_file = open('temp.txt', 'w')

    # Read the first record's description field.
    descr = coffee_file.readline()

    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

```

7.3 Processing Records

Modify a record in a file … **coffee.txt**

Program 7-18 *(cont.)*
(modify_coffee_records.py)



```
        # Write either this record to the temporary file,
        # or the new record if this is the one that is
        # to be modified.
        if descr == search:
            # Write the modified record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(new_qty) + '\n')

            # Set the found flag to True.
            found = True
        else:
            # Write the original record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(qty) + '\n')

        # Read the next description.
        descr = coffee_file.readline()

    # Close the coffee file and the temporary file.
    coffee_file.close()
    temp_file.close()

    # Delete the original coffee.txt file.
    os.remove('coffee.txt')

    # Rename the temporary file.
    os.rename('temp.txt', 'coffee.txt')

    # If the search value was not found in the file
    # display a message.
    if found:
        print 'The file has been updated.'
    else:
        print 'That item was not found in the file.'

# Call the main function.
main()
```

-37

7.3 Processing Records

Deleting a record in a file … **coffee.txt**

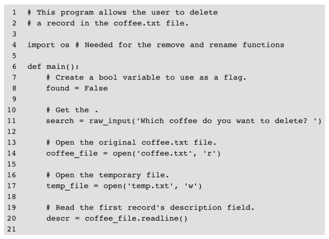
1. **Import `os` module**
2. **Set a flag to False**
3. **Get the search value**
4. **Open the original file for reading**
5. **Open a temporary file for writing**
6. **Read the first line from file**
7. **While NOT end-of-file**
   a. **Read coffee record**
   b. **Strip the newlines from each field**
   c. **Determine whether the record is a match to search value**
      i. **If match –Set flag to True**
      ii. **If NOT match – Write current record to temporary file**
   d. **Read the next line from the file**
8. **Close the original file**
9. **Close the temporary file**
10. **Delete original file**
11. **Rename temporary to the original file name**
12. **If flag is True – Display message "File updated"**
13. **If flag is False – Display message "Not Found"**

1-38

7.3 Processing Records

Deleting a record in a file … **coffee.txt**

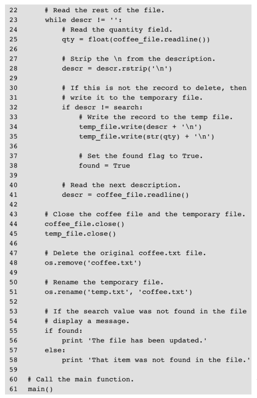
Program 7-19 (delete_coffee_record.py)



```
# This program allows the user to delete
# a record in the coffee.txt file.

import os # Needed for the remove and rename functions

def main():
    # Create a bool variable to use as a flag.
    found = False

    # Get the .
    search = raw_input('Which coffee do you want to delete? ')

    # Open the original coffee.txt file.
    coffee_file = open('coffee.txt', 'r')

    # Open the temporary file.
    temp_file = open('temp.txt', 'w')

    # Read the first record's description field.
    descr = coffee_file.readline()

```

1-39

Program 7-19 *(cont.)*
(delete_coffee_record.py)



```
    # Read the rest of the file.
    while descr != '':
        # Read the quantity field.
        qty = float(coffee_file.readline())

        # Strip the \n from the description.
        descr = descr.rstrip('\n')

        # If this is not the record to delete, then
        # write it to the temporary file.
        if descr != search:
            # Write the record to the temp file.
            temp_file.write(descr + '\n')
            temp_file.write(str(qty) + '\n')

            # Set the found flag to True.
            found = True

        # Read the next description.
        descr = coffee_file.readline()

    # Close the coffee file and the temporary file.
    coffee_file.close()
    temp_file.close()

    # Delete the original coffee.txt file.
    os.remove('coffee.txt')

    # Rename the temporary file.
    os.rename('temp.txt', 'coffee.txt')

    # If the search value was not found in the file
    # display a message.
    if found:
        print 'The file has been updated.'
    else:
        print 'That item was not found in the file.'

# Call the main function.
main()
```

1-40

7.4 Exceptions

Concept:

An exception is an error that occurs while a program is running, causing the program to abruptly halt. You can use the `try/exception` statement to gracefully handle exceptions.

1-41

7.4 Exceptions

- **Run time error**
- **Causes program to abruptly halt**
- **Error message is displayed …** ***traceback*** **– information regarding the cause of the exception**
- **For example:**

  ZeroDivisionError - division by zero
  IOError – file specified does not exist

1-42

7.4 Exceptions

ZeroDivisionError - division by zero

Program 7-20 (division.py)

```
# This program divides a number by another number.

def main():
    # Get two numbers.
    num1 = input('Enter a number: ')
    num2 = input('Enter another number: ')

    # Divide num1 by num2 and display the result.
    result = num1 / num2
    print num1, 'divided by', num2, 'is', result

# Call the main function.
main()
```

Program Output (with input shown in bold)

```
Enter a number: 10 [Enter]
Enter another number: 0 [Enter]
Traceback (most recent call last):
  File "C:/Python/division.py," line 13, in <module>
    main()
  File "C:/Python/division.py," line 9, in main
    result = num1 / num2
ZeroDivisionError: integer division or modulo by zero
```

7.4 Exceptions

IOError – file specified does not exist

Program 7-22 (display_file.py)

```
# This program displays the contents
# of a file.

def main():
    # Get the name of a file.
    filename = raw_input('Enter a filename: ')

    # Open the file.
    infile = open(filename, 'r')

    # Read the file's contents.
    contents = infile.read()

    # Display the file's contents.
    print contents

    # Close the file.
    infile.close()

# Call the main function.
main()
```

Program Output (with input shown in bold)

```
Enter a filename: bad_file.txt [Enter]
Traceback (most recent call last):
  File "C:/Python/display_file.py," line 21, in <module>
    main()
  File "C:/Python/display_file.py," line 9, in main
    infile = open(filename, 'r')
IOError: [Errno 2] No such file or directory: 'bad_file.txt'
```

7.4 Exceptions

- ***Exception handler*** **prevents the program from abruptly crashing**
- **Use the `try/except` statement**

```
try:
    statement      ⎫
    statement      ⎬— try block
    etc.           ⎭
except ExceptionName:———— except clause
    statement      ⎫
    statement      ⎬— handler
    etc.           ⎭
```

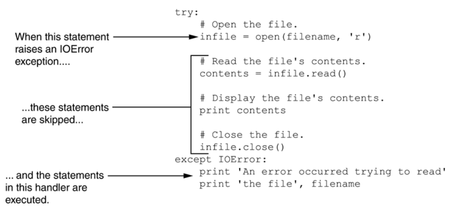
7.4 Exceptions

When `try/except` statement executes, the statements in the *try block* begin to execute:

- If a statement in the **try block** raises an exception that is specified by the ***ExceptionName*** in an **except clause**, then the **handler** that immediately follows the **except clause** executes. Then, the program resumes execution with the statement immediately following the `try/except` statement.
- If a statement in the **try block** raises an exception that is not specified by the ***ExecptionName*** in an **except clause**, then the program will halt with a traceback error message.
- If the statements in the **try block** execute without raising an exception, then any **except clauses** and **handlers** in the statement are skipped and the program resumes execution with the statements immediately following the `try/except` statement.

7.4 Exceptions

Figure 7-20 Sequence of events in the `try/except` statement


When this statement raises an IOError exception....
...these statements are skipped...
... and the statements in this handler are executed.

```
try:
    # Open the file.
    infile = open(filename, 'r')

    # Read the file's contents.
    contents = infile.read()

    # Display the file's contents.
    print contents

    # Close the file.
    infile.close()
except IOError:
    print 'An error occurred trying to read'
    print 'the file', filename
```

7.4 Exceptions

Program 7-23 (sales_report.py)

```
# This program displays the total of the
# amounts in the sales_data.txt file.

def main():
    # Initialize an accumulator.
    total = 0.0

    try:
        # Open the sales_data.txt file.
        infile = open('sales_data.txt', 'r')

        # Read the values from the file and
        # accumulate them.
        for line in infile:
            amount = float(line)
            total += amount

        # Close the file.
        infile.close()

        # Print the total.
        print 'Total: $%.2f' % total

    except IOError:
        print 'An error occured trying to read the file.'

    except ValueError:
        print 'Non-numeric data found in the file.'

    except:
        print 'An error occured.'

# Call the main function.
main()
```