

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

7. Files and Exceptions 299

7.1 Reading and Writing Text Files 300

7.1.1 Opening a File 300

Syntax 7.1 Opening and Closing Files 301

7.1.2 Reading from a File 301

7.1.3 Writing to a File 302

7.1.4 A File Processing Example 302

Self Check 303

Common Error 7.1 Backslashes in File Names 303

7.2 Text Input and Output 304

7.2.1 Iterating over the Lines of a File 304

Self Check 305

7.2.2 Reading Words 306

Self Check 308

7.2.3 Reading Characters 308

Self Check 309

7.2.4 Reading Records 309

Self Check 312

Special Topic 7.1 Reading the Entire File 312

Special Topic 7.2 Regular Expressions 312

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 316

SELF CHECK

• 1. Which of the following is the correct way to open the text file data.txt for reading?

theFile = file("data.txt", "w")
 theFile = open("data.txt", "r")
 theFile = open("data.txt", "w")
 theFile = file("data.txt", "r")

One correct, 0 errors, 100%

• 2. Answer the following true or false questions related to the use of text files.

True **False**

Executing the statement `infile = open("", "r")` will cause an error.
A run-time error occurs because the file will not exist.

True **False**

Executing the statement `infile = open("input.txt", "w")` will cause an error.
The file is opened for writing and *it is emptied*. The name of the file has no impact on the mode in which the file is opened.

True **False**

The following code will result in a run-time error:

```
infile = open("input.txt", "r")
line = infile.readline()
infile.close()
while line != "":
    print(line)
    line = infile.readline()
```

You cannot read from a closed file. The call to the close method should follow the loop, not precede it.

True **False**

The following code works correctly to write the string "Hello, World!" to the text file output.txt.
outfile = open("output.txt", "r")
outfile.write("Hello, World!\n")

The file was opened for reading, but it should have been opened for writing.

4 correct, 0 errors, 100%

• 3. Write statements to carry out the following tasks:

GOOD JOB!

Back to Page 303 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

7. Files and Exceptions 299

7.1 Reading and Writing Text Files 300

7.1.1 Opening a File 300

Syntax 7.1 Opening and Closing Files 301

7.1.2 Reading from a File 301

7.1.3 Writing to a File 302

7.1.4 A File Processing Example 302

Self Check 303

Common Error 7.1 Backslashes in File Names 303

7.2 Text Input and Output 304

7.2.1 Iterating over the Lines of a File 304

Self Check 305

7.2.2 Reading Words 306

Self Check 308

7.2.3 Reading Characters 308

Self Check 309

7.2.4 Reading Records 309

Self Check 312

Special Topic 7.1 Reading the Entire File 312

Special Topic 7.2 Regular Expressions 312

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 316

GOOD JOB! ✓

.. 3. Write statements to carry out the following tasks:

Task	Statement	Explanation
Open the text file <code>out.txt</code> for writing and assign the file object to the variable <code>outfile</code> .	<code>outfile = open("out.txt", "w")</code>	A file object represents a disk file.
Read the first line of text from the file referenced by <code>infile</code> and store it in variable <code>data</code> .	<code>data = infile.readline()</code>	To read a line of text from a text file, use the <code>readline</code> method.
Read an integer value from the next line of the file referenced by <code>infile</code> and store it in <code>value</code> .	<code>value = int(infile.readline())</code>	The <code>readline</code> method can only read strings. Numerical values must be converted using the <code>int</code> or <code>float</code> functions.
Close the file referenced by <code>theFile</code> .	<code>theFile.close()</code>	Be sure to close open files when you are done using them.
Write the word <code>Hello</code> to the file referenced by <code>out</code> .	<code>out.write("Hello")</code>	To write to a file, use the <code>write</code> method.
Close the output file referenced by <code>out</code> .	<code>out.close()</code>	If you forgot to close an output file, some output may not be written to the disk file.

6 correct, 0 errors, 100%

Start over

GOOD JOB! ✓

.. 4. Rearrange the following lines to produce a program that reads and prints the contents of a file to the terminal, one line at a time, with the lines numbered, 1., 2., 3., and so on.

```
inFile = open("input.txt", "r")
counter = 0
line = inFile.readline()
while line != "":
    counter = counter + 1
    print("%d. %s" % (counter, line))
    line = inFile.readline()
inFile.close()
```

8 correct, 0 errors, 100%

Start over

.. 5. Modify the `total.py` program so that it also prints the largest and smallest values, not just the total and average. For example, the output for the sample input in Section 7.1.4 would look as follows:

```
32.00
54.00
67.50
29.00
35.00
38.25
115.00
```

Back to Page 302 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

7. Files and Exceptions 299

7.1 Reading and Writing Text Files 300

7.1.1 Opening a File 300

Syntax 7.1 Opening and Closing Files 301

7.1.2 Reading from a File 301

7.1.3 Writing to a File 302

7.1.4 A File Processing Example 302

Self Check 303

Common Error 7.1 Backslashes in File Names 303

7.2 Text Input and Output 304

7.2.1 Iterating over the Lines of a File 304

Self Check 305

7.2.2 Reading Words 306

Self Check 308

7.2.3 Reading Characters 308

Self Check 309

7.2.4 Reading Records 309

Self Check 312

Special Topic 7.1 Reading the Entire File 312

Special Topic 7.2 Regular Expressions 312

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 319

How To 7.1 Processing Text Files 319

Worked Example 7.1 Analyzing Baby Names 322

Toolbox 7.2 Working with Files and Directories 325

Start over

• 5. Modify the total.py program so that it also prints the largest and smallest values, not just the total and average. For example, the output for the sample input in Section 7.1.4 would look as follows:

```
32.00
54.00
67.50
29.00
35.00
80.25
115.00
.....
Total: 412.75
Average: 58.96
Max: 115.00
Min: 29.00
```

total2.py

```
1# This program reads a file containing numbers and writes the numbers to another file, lined up in a column and followed by their total and average.
4
5# Prompt the user for the name of the input and output files.
7inputFileName = input("Input file name: ")
8outputFileName = input("Output file name: ")
9
10# Open the input and output files.
11infile = open(inputFileName, "r")
12outfile = open(outputFileName, "w")
13
14# Read the input and write the output.
15total = 0.0
16count = 0
17list = []
18line = infile.readline()
19while line != "":
20    value = float(line)
21    list.append(value)
22    outfile.write("%15.2f\n" % value)
23    total += value
24    count += 1
25    line = infile.readline()
26largest = max(list)
27smallest = min(list)
28
29# Output the total, average, max and min
30outfile.write("%15.2f\n" % total)
31outfile.write("%15.2f\n" % average)
32outfile.write("Max: %15.2f\n" % largest)
33outfile.write("Min: %15.2f\n" % smallest)
4
4# Close the files.
4infile.close()
4outfile.close()
```

CodeCheck Reset

Running total2.py

Test 1

Input file name: `input.txt`
Output file name: `report.txt`

report.txt:

```
32.00
54.00
67.50
29.00
35.00
80.25
115.00
.....
Total: 412.75
Average: 58.96
Max: 115.00
Min: 29.00
```

pass

Score

1/1



	Python For Everyone, Enhanced eText	...
Cay S. Horstmann; Rance D. Necaise		
Expand Collapse		
Chapter Summary 180		
Interactive Review and Practice 182		
End-of-Chapter Exercises EX4-1		
5. Functions 183		
6. Lists 245		
6.1 Basic Properties of Lists 246		
6.2 List Operations 252		
6.3 Common List Algorithms 259		
6.4 Using Lists with Functions 268		
6.5 Problem Solving: Adapting Algorithms 275		
6.6 Problem Solving: Discovering Algorithms by Manipulating Physical Objects 282		
6.7 Tables 285		
Chapter Summary 296		
Interactive Review and Practice 298		
End-of-Chapter Exercises EX6-1		
7. Files and Exceptions 299		
8. Sets and Dictionaries 357		
9. Objects and Classes 393		
10. Inheritance 443		

SELF CHECK

• 1. The following function is supposed to add 1 to every element in a list of integers.

```
def addOne(values) :  
    for i in range(len(values)) :  
        values[i] = values[i] + 1
```

What is wrong with the function?

The statement `print(values)` must be added to the end of the function.
 The statement `return values` must be added to the end of the function.
 The for loop must be replaced with a while loop.
 There is nothing wrong with the function. It works as intended.

One correct, 0 errors, 100%

• 2. What output is generated by the program below?

```
def main() :  
    scores = [45.6, 67.8, 89.4]  
    addBonus(scores, 3.0)  
    print(scores[2])  
  
def addBonus(values, bonus) :  
    for k in range(len(values)) :  
        values[k] = values[k] + bonus  
  
main()
```

67.8
 70.8
 89.4
 92.4

One correct, 0 errors, 100%

•• 3. Your task is to design a function

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

7. Files and Exceptions 299

7.1 Reading and Writing Text Files 300

7.1.1 Opening a File 300

Syntax 7.1 Opening and Closing Files 301

7.1.2 Reading from a File 301

7.1.3 Writing to a File 302

7.1.4 A File Processing Example 302

Self Check 303

Common Error 7.1 Backslashes in File Names 303

7.2 Text Input and Output 304

7.2.1 Iterating over the Lines of a File 304

Self Check 305

7.2.2 Reading Words 306

Self Check 308

7.2.3 Reading Characters 308

Self Check 309

7.2.4 Reading Records 309

Self Check 312

Special Topic 7.1 Reading the Entire File 312

Special Topic 7.2 Regular Expressions 312

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 319

How To 7.1 Processing Text Files 319

Worked Example 7.1 Analyzing Baby Names 322

Toolbox 7.2 Working with Files and Directories 325

SELF CHECK

1. If the text file input.txt contains the following contents

```
apple
pear
banana
```

what is printed by the following code segment?

```
infile = open("input.txt", "r")
for word in infile :
    word = word.rstrip()
    print(word)
```

apple pear banana

apple
pear
banana

apple
pear
banana

apple, pear, banana

One correct, 0 errors, 100%

2. Rearrange the following lines to produce a program that reads a text file containing exam grades, stored one per line, and prints the highest exam grade.

GOOD JOB! ✓

```
infile = open("grades.txt", "r")
maxGrade = 0
for line in infile :
    grade = int(line)
    if grade > maxGrade :
        maxGrade = grade
print("Highest grade =", maxGrade)
infile.close()
```

8 correct, 0 errors, 100%, 74 seconds

Start over



Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise

[Expand](#) | [Collapse](#)

- ▲ 7. Files and Exceptions 299
- ▲ 7.1 Reading and Writing Text Files 300
 - 7.1.1 Opening a File 300
 - Syntax 7.1 Opening and Closing Files 301
 - 7.1.2 Reading from a File 301
 - 7.1.3 Writing to a File 302
 - 7.1.4 A File Processing Example 302
- Self Check 303
- Common Error 7.1 Backslashes in File Names 303
- ▲ 7.2 Text Input and Output 304
 - 7.2.1 Iterating over the Lines of a File 304
 - Self Check 305
- 7.2.2 Reading Words 306
 - Self Check 308
 - 7.2.3 Reading Characters 308
 - Self Check 309
 - 7.2.4 Reading Records 309
 - Self Check 312
 - Special Topic 7.1 Reading the Entire File 312
 - Special Topic 7.2 Regular Expressions 312
 - Special Topic 7.3 Character Encodings 313
 - Toolbox 7.1 Working with CSV Files 314
- ▲ 7.3 Command Line Arguments 316
 - Self Check 319
 - How To 7.1 Processing Text Files 319
 - Worked Example 7.1 Analyzing Baby Names 322
 - Toolbox 7.2 Working with Files and Directories 325

Table 4 String Splitting Examples

Statement	Result	Comment
<code>string = "a,b,c,d"</code> <code>string.split(",")</code>	<code>"a" "b" "c" "d"</code>	The string is split at each comma.
<code>string = "a b c"</code> <code>string.split()</code>	<code>"a" " " "c"</code>	The string is split using the blank space as the delimiter. Consecutive blank spaces are treated as one space.
<code>string = "a b c"</code> <code>string.split(" ")</code>	<code>"a" "b" " " "c"</code>	The string is split using the blank space as the delimiter. With an explicit argument, the consecutive blank spaces are treated as separate delimiters.
<code>string = "a:b:c:d"</code> <code>string.split(":", 1)</code>	<code>"a:" "bc:d"</code>	The string is split into 2 parts starting from the front. The split is made at the first colon.
<code>string = "a:b:c:d"</code> <code>string.rsplit(":", 1)</code>	<code>"a:bc" "d"</code>	The string is split into 2 parts starting from the end. The split is made at the last colon.

SELF CHECK

3. What is printed by the following code segment, if the file object `infile` refers to a file that contains one line of text: 1600 Pennsylvania Avenue NW, Washington, DC 20500?

```
count = 0
line = infile.readline()
line = line.rstrip()
wordList = line.split()
for word in wordList :
    count = count + 1
print(count)
```

6
 7
 8
 9

One correct, 0 errors, 100%

4. Rearrange the following lines to produce statements that read a text file and print all words, suppressing adjacent duplicates. For example, if the file contains two words "the" with no words between them, only one is printed.

GOOD JOB! ✓

```
infile = open("input.txt", "r")
previous = ""
for line in infile :
    line = line.strip()
    wordList = line.split()
    for word in wordList :
        if word != previous :
            print(word, end="")
        previous = word
infile.close()
```

10 correct, 0 errors, 100%, 152 seconds

[Start over](#)

The screenshot shows a chapter from "Python For Everyone, Enhanced eText" by Cay S. Horstmann and Rance D. Necaise. The chapter is titled "7. Files and Exceptions". A "SELF CHECK" section is open, containing two questions:

• 5. Given a text file quote.txt that contains the following text:
Home computers are being called upon to perform many new functions, including the consumption of homework formerly eaten by the dog. -Doug Larson
What is printed by the following code segment?

```
letterCounts = [0] * 26
inputFile = open('quote.txt', 'r')
char = inputFile.read(1)
while char != '':
    char = char.upper()
    if 'A' <= char <= 'Z':
        code = ord(char) - ord('A')
        letterCounts[code] = letterCounts[code] + 1
    char = inputFile.read(1)

inputFile.close()
print(letterCounts[1])
```

Four options are provided: 5, 0, 2, and 1.

One correct, 0 errors, 100%

• 6. Write a program that reads a file, one character at a time, and prints out how many uppercase characters it contains and how many lowercase characters it contains.

```
countchars.py
1 numUppercase = 0
2 numLowercase = 0
3
4 filename = input("Input filename: ")
5 infile = open(filename, "r")
6
7 file = infile.read()
8 for char in file:
9     if char.isupper():
10         numUppercase = numUppercase + 1
11     elif char.islower():
12         numLowercase = numLowercase + 1
13
14 infile.close()
15
16 print("# of uppercase characters:", numUppercase)
17 print("# of lowercase characters:", numLowercase)
```

CodeCheck Reset

Running countchars.py

Test 1
Input filename: in1.txt
of uppercase characters: 42
of lowercase characters: 1574

pass
Test 2
Input filename: in2.txt
of uppercase characters: 11
of lowercase characters: 917

pass
Score
2/2



Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise

[Expand](#) | [Collapse](#)

- [7. Files and Exceptions](#) 299
- [7.1 Reading and Writing Text Files](#) 300
 - [7.1.1 Opening a File](#) 300
 - [Syntax 7.1 Opening and Closing Files](#) 301
 - [7.1.2 Reading from a File](#) 301
 - [7.1.3 Writing to a File](#) 302
 - [7.1.4 A File Processing Example](#) 302
- [Self Check](#) 303
- [Common Error 7.1 Backslashes in File Names](#) 303
- [7.2 Text Input and Output](#) 304
 - [7.2.1 Iterating over the Lines of a File](#) 304
 - [Self Check](#) 305
 - [7.2.2 Reading Words](#) 306
 - [Self Check](#) 308
 - [7.2.3 Reading Characters](#) 308
 - [Self Check](#) 309
- [7.2.4 Reading Records](#) 309
 - [Self Check](#) 312
 - [Special Topic 7.1 Reading the Entire File](#) 312
 - [Special Topic 7.2 Regular Expressions](#) 312
 - [Special Topic 7.3 Character Encodings](#) 313
 - [Toolbox 7.1 Working with CSV Files](#) 314
- [7.3 Command Line Arguments](#) 316
 - [Self Check](#) 319
 - [How To 7.1 Processing Text Files](#) 319
 - [Worked Example 7.1 Analyzing Baby Names](#) 322
- [Toolbox 7.2 Working with Files and Directories](#) 325

```

1 # Open the file for reading.
2 with open('itemlist.txt') as infile:
3     # Extract the two data fields.
4     for line in infile:
5         parts = line.split(',')
6         item = parts[0]
7         price = float(parts[1])
8
9         # Increment the total.
10        total = total + price
11
12        # Write the output.
13        outfile.write("%-20s%10.2f\n" % (item, price))
14
15 # Write the total price.
16 outfile.write("%-20s%10.2f\n" % ("Total:", total))
17
18 # Close the files.
19 infile.close()
20 outfile.close()

```

[Input](#)

```

1 itemlist.txt
2 out.txt
3

```

[Run](#) [Reset](#)

 **SELF CHECK**

• 7. What is printed by the following code segment?

```

line = "hello world!"
parts = line.split()
print(parts)

```

hello,world!
 hello world!
 "hello", "world!"
 ["hello", "world!"]

One correct, 0 errors, 100%

• 8. Suppose the input file contains the single line of text

```

6E4 6,995.00

```

What are the values of each of the expressions after the following code segment is executed? If an error occurs, enter an **error** for the value.

```

line = infile.readline()
parts = line.split()
x1 = float(parts[0])
x2 = float(parts[1])

```

GOOD JOB! ✓

Expression	Value	Explanation
<code>len(parts)</code>	2	The string "6E4" is a scientific notation for that value.
<code>x1</code>	60000	
<code>x2</code>	error	The last statement will cause a <code>ValueError</code> exception because a comma is not considered part of a floating-point number in Python.

3 correct, 0 errors, 100%

[Start over](#)

7.2.4

7.2.4 - Reading Records (7. 8)

309 / 554



[Expand](#) | [Collapse](#)

Self Check	303
Common Error 7.1 Backslashes in File Names	303
7.2 Text Input and Output	304
7.2.1 Iterating over the Lines of a File	304
Self Check	305
7.2.2 Reading Words	306
Self Check	308
7.2.3 Reading Characters	308
Self Check	309
7.2.4 Reading Records	309
Self Check	312
Special Topic 7.1 Reading the Entire File	312
Special Topic 7.2 Regular Expressions	312
Special Topic 7.3 Character Encodings	313
Toolbox 7.1 Working with CSV Files	314
7.3 Command Line Arguments	316
Self Check	319
How To 7.1 Processing Text Files	319
Worked Example 7.1 Analyzing Baby Names	322
Toolbox 7.2 Working with Files and Directories	325



FILE CHECK

- 1. Select the output produced by the program below, assuming that the program is executed from the command line as follows: `python question.py -h datafile.txt results.txt`

```
from sys import argv  
print(argv[1], argv[2])
```

- datafile.txt results.txt
 - h datafile.txt
 - question.py -h
 - Execution of the program would result in a run-time "index of out bounds error".

One correct, 0 errors, 100%

- 2. Assume that a program is executed as

```
python search.py -min 90 -d grades.txt
```

What are the values of the following expressions? (You do not need to enclose strings in quotes.)

GOOD JOB! ✓

Expression	Value	Explanation
<code>len(argv)</code>	5	This call consists of the program name and four command-line arguments. The command name <code>python</code> is not part of the <code>argv</code> list, but the program name <code>search.py</code> is.
<code>argv[1]</code>	<code>-min</code>	By convention, options start with a dash.
<code>int(argv[2])</code>	90	This is the value of the <code>-min</code> option.
<code>argv[3]</code>	<code>-d</code>	This is an option without a value, or a Boolean option. Its presence changes the program behavior.
<code>argv[4]</code>	<code>grades.txt</code>	Because this argument doesn't start with a dash, it is most likely a file name. It is entirely up to the program how to interpret the command-line arguments.

5 correct, 0 errors, 100%

[Start over](#)

- 3. Suppose the search.py program is started with the command

```
python search.py -max 20 student.txt result.txt
```

Walk through the following code that processes the command-line arguments. You do not need to enter quotation marks around string values in this exercise.

Python For Everyone, Enhanced eText

Cay S. Horstmann, Rance D. Necaise

Expand | Collapse

Self Check 303

Common Error 7.1 Backslashes in File Names 303

7.2 Text Input and Output 304

7.2.1 Iterating over the Lines of a File 304

Self Check 305

7.2.2 Reading Words 306

Self Check 308

7.2.3 Reading Characters 308

Self Check 309

7.2.4 Reading Records 309

Self Check 312

Special Topic 7.1 Reading the Entire File 312

Special Topic 7.2 Regular Expressions 312

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 319

How To 7.1 Processing Text Files 319

Worked Example 7.1 Analyzing Baby Names 322

Toolbox 7.2 Working with Files and Directories 325

Computing & Society 7.1 Encryption Algorithms 327

7.4 Binary Files and Random Access (Optional) 328

7.4.1 Reading and Writing Binary Files 328

7.4.2 Random Access 329

7.4.3 Image Files 330

7.4.4 Processing BMP Files 331

Self Check 334

5 correct, 0 errors, 100%

Start over

*** 3. Suppose the search.py program is started with the command
`python search.py -max 20 student.txt result.txt`

Walk through the following code that processes the command-line arguments. You do not need to enter quotation marks around string values in this exercise.

GOOD JOB! ✓

i	arg	min	max	infile	outfile
1	-max				
2			20		
3	student.txt			student.txt	
4	result.txt				result.txt

```
from sys import argv
min = -1
max = 1
infile = ""
outfile = ""
i = 1
while i < len(argv) :
    arg = argv[i]
    if arg.startswith("-") :
        if arg == "-min" :
            i = i + 1
            min = int(argv[i])
        elif arg == "-max" :
            i = i + 1
            max = int(argv[i])
        else :
            usage()
    elif infile == "" :
        infile = arg
    elif outfile == "" :
        outfile = arg
    else :
        usage()
    i = i + 1
```

22 correct, 0 errors, 100%

Start over

*** 4. Complete the following function that parses the command line arguments for a program that can be used to copy a text file and returns the argument data in a list. The function should assume the program requires two arguments and allows for one optional argument:
 ◦ -d adds a blank line between the lines in the destination file. The default is single-spaced lines.

For example,

```
python filecopy.py report.txt -d reportdb1.txt
```

Copies the text file report.txt to reportdb1.txt with the lines double spaced. Assume the first non-switch argument is the source file name and the second non-switch argument is the destination file name. If a sufficient number of arguments are not provided or an invalid switch is provided, the function should return an empty list.

```
parseArgs
1## 2 Parses the command-line arguments and returns the data in a list. The
2# program requires two arguments and allows for one optional argument.
3# program arg list of command-line arguments passed to the program
4# program arg list of command-line arguments passed to the program
5# program arg list of command-line arguments passed to the program
6# where srcfile and destfile are strings and dblspace is a Boolean. If
7# a sufficient number of arguments are not supplied or an invalid
8# switch is provided, an empty list is returned.
9#
10def parseArgs(args):
11    """
```

CodeCheck Reset

Score: 0

Back to Page

316 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann, Rance D. Necaise

Expand | Collapse

Special Topic 7.3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 319

How To 7.1 Processing Text Files 319

Worked Example 7.1 Analyzing Baby Names 322

Toolbox 7.2 Working with Files and Directories 325

Computing & Society 7.1 Encryption Algorithms 327

7.4 Binary Files and Random Access (Optional) 328

7.4.1 Reading and Writing Binary Files 328

7.4.2 Random Access 329

7.4.3 Image Files 330

7.4.4 Processing BMP Files 331

Self Check 334

Worked Example 7.2 Graphics: Displaying a Scene File 334

7.5 Exception Handling 337

7.5.1 Raising Exceptions 338

Syntax 7.2 Raising an Exception 339

7.5.2 Handling Exceptions 339

Syntax 7.3 Handling Exceptions 341

SELF CHECK

1. Which of the following is *not* a characteristic of binary files?

- Saves space.
- Allows for random access.
- Data items are stored in human readable form.
- Data items are stored as binary values.

One correct, 0 errors, 100%

2. If a byte consists of 8 bits, what are the minimum and maximum values of one byte?

- 0 ... 255
- 1 ... 256
- 0 ... 256
- 1 ... 255

One correct, 0 errors, 100%

3. Rearrange the following lines of code to produce code that swaps the values at positions pos1 and pos2 in a file.

GOOD JOB! ✓

```
infile = open(filename, "rb")
infile.seek(pos2)
value2 = infile.read(4)
infile.seek(pos1)
value1 = infile.read(4)
infile.seek(pos1)
infile.write(value2)
infile.seek(pos2)
infile.write(value1)
```

5 correct, 0 errors, 100%

Start over

Back to Page / 331 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Special Topic / 3 Character Encodings 313

Toolbox 7.1 Working with CSV Files 314

7.3 Command Line Arguments 316

Self Check 319

How To 7.1 Processing Text Files 319

Worked Example 7.1 Analyzing Baby Names 322

Toolbox 7.2 Working with Files and Directories 325

Computing & Society 7.1 Encryption Algorithms 327

7.4 Binary Files and Random Access (Optional) 328

7.4.1 Reading and Writing Binary Files 328

7.4.2 Random Access 329

7.4.3 Image Files 330

7.4.4 Processing BMP Files 331

Self Check 334

Worked Example 7.2 Graphics: Displaying a Scene File 334

7.5 Exception Handling 337

7.5.1 Raising Exceptions 338

Syntax 7.2 Raising an Exception 339

```
infile = open(filename, "wb")
infile.seek(pos2)
value2 = infile.read(4)
infile.seek(pos1)
value1 = infile.read(4)
infile.seek(pos1)
infile.write(value2)
infile.seek(pos2)
infile.write(value1)
```

5 correct, 0 errors, 100%

Start over

.. 4. Indicate whether each of the following statements is true or false.

✓ True False

The following code segment replaces the last byte of a file with 0.

```
binFile.seek(0)
binFile.write(bytes([0]))
```

It replaces the first byte of a file with 0.

✓ True False

The image processing program could have been implemented using sequential file access only in which the file was opened for reading and without creating a new file.

We could have read the header values and pixel data sequentially, but to update the pixels, we had to move the file marker backward to overwrite the original values.

✓ True False

If you run the `imageproc.py` program twice on the same image file, you get the original image back.

Applying the negative filter to an image creates a new image in which each pixel is inverted. When the inverted pixel is inverted a second time, it results in the original. This is similar to inverting a Boolean value. If you invert False, the result is True. If you invert True, the result is False.

✓ True False

If a BMP file stores a 100×100 pixel image in BMP format, with the image data starting at offset 64, the total file size is 30,064 bytes.

We need 3×100 bytes for each row. There is no padding because this number is divisible by 4. The total size = $3 \times 100 \times 100 + 64 = 30,064$ bytes.

4 correct, 0 errors, 100%

331 / 554

Python For Everyone, Enhanced eText

Cay S. Horstmann, Rance D. Necaise

Expand | Collapse

WORKING WITH FILES AND DIRECTORIES 340

Computing & Society 7.1 Encryption Algorithms 327

7.4 Binary Files and Random Access (Optional) 328

 7.4.1 Reading and Writing Binary Files 328

 7.4.2 Random Access 329

 7.4.3 Image Files 330

 7.4.4 Processing BMP Files 331

Self Check 334

Worked Example 7.2 Graphics: Displaying a Scene File 334

7.5 Exception Handling 337

 7.5.1 Raising Exceptions 338

 Syntax 7.2 Raising an Exception 339

 7.5.2 Handling Exceptions 339

 Syntax 7.3 Handling Exceptions 341

 7.5.3 The finally Clause 341

Syntax 7.4 The finally Clause 342

Self Check 342

Programming Tip 7.1 Raise Early, Handle Late 342

Programming Tip 7.2 Do Not Use except and finally in the Same try Statement 342

Special Topic 7.4 The with Statement 343

SELF CHECK

• 1. Complete the code fragment below, which is designed to raise an exception with an appropriate error message when variable index is negative.

```
if index < 0 :  
    _____  
  
 raise IndexError  
 raise IndexError("Negative index")  
 except IndexError  
 except IndexError as exception
```

One correct, 0 errors, 100%

• 2. Consider the following function

```
##  
# Returns the last n characters from a given string.  
# @param string the given string  
# @param n the number of characters to take from the tail end of string  
# @return a string consisting of the last n characters of string  
#  
def tail(string, n) :
```

Rearrange the following lines of code to produce the function body, raising the appropriate exception if argument n is out of range.

GOOD JOB! ✓

```
if n < 0 or n > len(string) :  
    raise IndexError("n out of bounds")  
newStr = ""  
for i in range(len(string) - n, len(string)) :  
    newStr = newStr + string[i]  
return newStr
```

6 correct, 0 errors, 100%

Start over

• 3. In the preceding problem, the tail function raised an exception when the arguments were invalid. Suppose the body of the function did not include the raise statement and it was called as tail("test", -2). What would

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

6 correct, 0 errors, 100%

[Start over](#)

• 3. In the preceding problem, the tail function raised an exception when the arguments were invalid. Suppose the body of the function did not include the raise statement and it was called as tail("test", -2). What would have happened?

The function would have raised a `TypeError`.
 The function would have raised an `IndexError` with the message `n out of bounds`.
 The function would have raised an `IndexError` with a less informative message.
 The function would have returned the empty string.

One correct, 0 errors, 100%

•• 4. This walkthrough shows how an exception is handled. The `int` function raises a `ValueError` exception when it is called with a string that does not contain the digits of an integer. Assume that the code segment reads the input

42
five

GOOD JOB! ✓

result	total	inputStr
0	0	"42"
42	42	"five"
-1		

```
result = 0
total = 0
while result != -1 :
    inputStr = input("Enter an integer: ")
    try :
        result = int(inputStr)
        total = total + result
    except ValueError :
        result = -1
```

4 correct, 0 errors, 100%

[Start over](#)

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Syntax 7.3 Handling Exceptions 341

7.5.3 The finally Clause 341

Syntax 7.4 The finally Clause 342

Self Check 342

Programming Tip 7.1 Raise Early, Handle Late 342

Programming Tip 7.2 Do Not Use except and finally in the Same try Statement 342

Special Topic 7.4 The with Statement 343

Toolbox 7.3 Reading Web Pages 343

7.6 Application: Handling Input Errors 344

Self Check 347

Toolbox 7.4 Statistical Analysis 348

Worked Example 7.3 Creating a Bubble Chart 352

Computing & Society 7.2 The Ariane Rocket Incident 355

Chapter Summary 356

Interactive Review and Practice 356

End-of-Chapter Exercises EX7-1

8. Sets and Dictionaries 357

9. Objects and Classes 393

10. Inheritance 443

11. Recursion 489

12. Sorting and Searching 525

Appendices A-1

Glossary R-1

SELF CHECK

1. Consider the following code segment that reads three integer values from a text file. What type of exception is raised if the file only contains two values?

```
infile = open("values.txt", "r")
value1 = int(infile.readline())
value2 = int(infile.readline())
value3 = int(infile.readline())
```

TypeError
 ArithmeticError
 RuntimeError
 ValueError

One correct, 0 errors, 100%

2. Consider the following function

```
# Returns an integer value read from the user that is within a given range.
# @param low the low end of the valid range
# @param high the high end of the valid range
# @return the valid integer read from the user
#
def readFromUser(low, high) :
```

Rearrange the following lines of code for the body of the function that continues to read data from the user until an integer in the given range is read. Use all lines in your solution.

GOOD JOB! ✓

```
done = False
while not done :
    try :
        value = int(input("Enter an integer value: "))
        if value < low or value > high :
            print("Error: value is out of range.")
        else :
            done = True
    except ValueError :
        print("Error: invalid data type.")
return value
```

11 correct, 0 errors, 100%

Start over

Back to Page

344 / 554

< Q A ⌂ ...

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Syntax 7.3 Handling Exceptions	341
7.5.3 The finally Clause	341
Syntax 7.4 The finally Clause	342
Self Check	342
Programming Tip 7.1 Raise Early, Handle Late	342
Programming Tip 7.2 Do Not Use except and finally in the Same try Statement	342
Special Topic 7.4 The with Statement	343
Toolbox 7.3 Reading Web Pages	343
7.6 Application: Handling Input Errors	344
Self Check	347
Toolbox 7.4 Statistical Analysis	348
Worked Example 7.3 Creating a Bubble Chart	352
Computing & Society 7.2 The Ariane Rocket Incident	355

11 correct, 0 errors, 100%

Start over

•• 3. Consider the [analyzedata.py](#) program and indicate whether each of the following statements is true or false.

True **False**

The `readFile` function should handle `IOError` exceptions itself.
To make the function as portable as possible, the exceptions are better handled in the `main` function.

True **False**

If the user specifies a file that exists, but is empty, the call `value = int(line)` raises a `ValueError` exception which propagates back to `main`, where it is handled.
An error message is then printed and the user can specify another file.

True **False**

The `readData` function should check the input line to ensure that a `ValueError` exception is not raised when there are not enough values in the file.
We want that exception raised and allowed to propagate back so that someone else can handle the problem of a bad data file.

True **False**

Consider the `try/finally` statement in the `readFile` function. The call to the `open` function can be placed inside the `try` block with the same results.
If it is opened inside the `try` block and an exception is raised when opening the file, the `infile` variable will not be defined (due to the error), but the `finally` clause will attempt to use the undefined `infile` variable.

4 correct, 0 errors, 100%

Back to Page 344 / 554 >