Chapter 6

Files and Exceptions I

THE PRACTICE OF COMPUTING USING

PYTHON

3RD EDITION

WILLIAM RICHARD
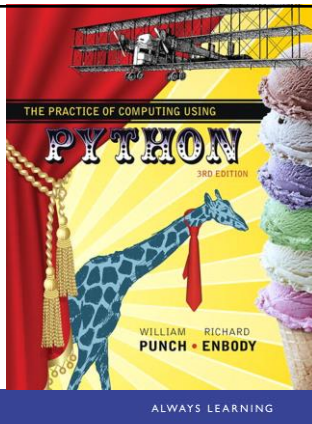**PUNCH • ENBODY**

PEARSON   ALWAYS LEARNING

What is a file?

- a file is a collection of data that is stored on secondary storage like a disk or a thumb drive
- accessing a file means establishing a connection between the file and the program and moving data between the two

---

Two Types of Files

files come in two general types:
- *text files* – files where control characters such as `"\n"` are translated
  – these files are generally human readable
- *binary files* – all the information is taken directly without translation
  – not readable and contains non-readable info

File Object or Stream

- when opening a file, you create a file object or file stream that is a connection between the file information on disk and the program
- the stream contains a buffer of the information from the file, and provides the information to the program

---
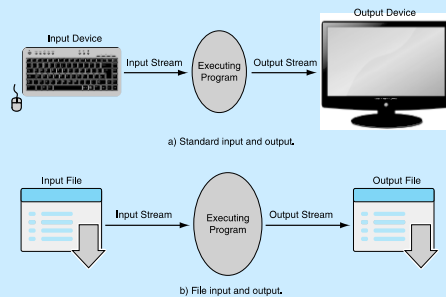
Figure 6.1 Input-output streams.

Buffering

- reading from a disk is very slow
- thus the computer will read a lot of data from a file in the hopes that, if you need the data in the future, it will be buffered in the file object
  – this means that the file object contains a copy of information from the file called a cache (pronounced "cash")

Making a File Object

```
my_file = open("my_file.txt", "r")
```

- **my_file** is the file object, which contains the buffer of information
- the open function creates the connection between the disk file and the file object
- the first quoted string is the file name on disk, the second is the mode to open it (here, **"r"** means to read)

Where is the Disk File?

- when opened, the name of the file can come in one of two forms:
  - **"file.txt"** assumes the file name is file.txt and that it is located in the current program directory
  - **"c:\bill\file.txt"** is the fully qualified file name and includes the directory path

Different Modes

Mode	How Opened	File Exists	File Does Not Exist
'r'	read-only	Opens that file	Error
'w'	write-only	Clears the file contents	Creates and opens a new file
'a'	write-only	File contents left intact and new data appended at file's end	Creates and opens a new file
'r+'	read and write	Reads and overwrites from the file's beginning	Error
'w+'	read and write	Clears the file contents	Creates and opens a new file
'a+'	read and write	File contents left intact and read and write at file's end	Creates and opens a new file

Table 6.1 File modes.

Be Careful with Write Modes

- be careful if you open a file with the **'w'** mode – it sets an existing file's contents to be empty, destroying any existing data
- the **'a'** mode is nicer, allowing you to write to the end of an existing file without changing the existing contents

Text Files Use Strings

- if you are interacting with text files (which is all we will do in this book), remember that *everything is a string*
  - everything read is a string
  - if you write to a file, you can only write a string

Writing to a File

once you have created a file object, opened for reading, you can use the print command

- add **file=fileobject** to the print command

```
# open file for writing:
#   creates file if it does not exist
#   overwrites file if it exists
>>> temp_file = open("temp.txt","w")
>>> print("first line", file=temp_file)
>>> print("second line", file=temp_file)
>>> temp_file.close()
```
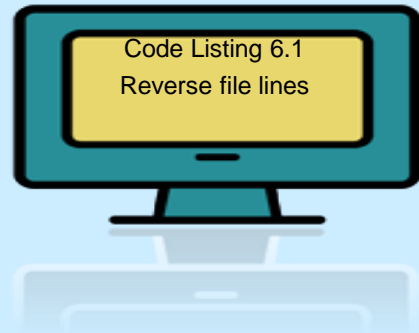
Closing the File

when the program is finished with a file, we must **close** it

- flush the buffer contents from the computer to the file
- tear down the connection to the file
- **close** is a method of a file obj
  file_obj.close()
- all files should be closed!

Code Listing 6.1
Reverse file lines

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")

for line_str in input_file:
    new_str = ''
    line_str = line_str.strip()        # get rid of carriage return
    for char in line_str:
        new_str = char + new_str       # concat at the left (reverse)
    print(new_str,file=output_file)    # print to output_file

    # include a print to shell so we can observe progress
    print('Line: {:12s} reversed is: {:s}'.format(line_str, new_str))
input_file.close()
output_file.close()
```

Word Puzzle

the following listings show how one might solve the following puzzle: look through a file of words, one word per line, and identify any word that has all the vowels in order, with only one example of each vowel

- for example, "facetious"

Code Listing 6.3
clean_word

```
def clean_word(word):
    """Return word in lower case stripped of whitespace."""
    return word.strip().lower()
```

Code Listing 6.4

```python
data_file = open("dictionary.txt", "r")

def clean_word(word):
    """Return Word in lower case stripped of whitespace."""
    return word.strip().lower()

# main program
for word in data_file:          # for each word in the file
    word = clean_word(word)     # clean the word
    if len(word) <= 6:          # if word is too small, skip it
        continue
print(word)
```

Code Listing 6.5
get_vowels

```python
def get_vowels_in_word(word):
    """Return vowels in string word--include repeats."""
    vowel_str = "aeiou"
    vowels_in_word = ""
    for char in word:
        if char in vowel_str:
            vowels_in_word += char
    return vowels_in_word
```

Code Listing 6.6
Full Solution

```python
# Find a word with a single example of the vowels a, e, i, o, u in that order

data_file = open("dictionary.txt", "r")

def clean_word(word):
    """Return word in lower case stripped of whitespace."""
    return word.strip().lower()

def get_vowels_in_word(word):
    """Return vowels in string word--include repeats."""
    vowel_str = "aeiou"
    vowels_in_word = ""
    for char in word:
        if char in vowel_str:
            vowels_in_word += char
    return vowels_in_word

# main program
print("Find words containing vowels 'aeiou' in that order:")
for word in data_file:          # for each word in the file
    word = clean_word(word)     # clean the word
    if len(word) <= 6:          # if word is too small, skip it
        continue
    vowel_str = get_vowels_in_word(word)  # get vowels in word
    if vowel_str == 'aeiou':              # check all vowels in order
        print(word)
```

Exceptions
First Cut

How to Deal with Problems

- most modern languages provide methods to deal with 'exceptional' situations
- gives the programmer the option to keep the user from having the program stop without warning
- again, this is not about fundamental CS, but about doing a better job as a programmer

---

What Counts as Exceptional

- errors – indexing past the end of a list, trying to open a nonexistent file, fetching a nonexistent key from a dictionary, etc.
- events – search algorithm doesn't find a value (not really an error), mail message arrives, queue event occurs, mouse button is clicked

---

Exceptions (2)

- ending conditions – file should be closed at the end of processing, list should be sorted after being filled
- weird stuff – for rare events, keep from clogging your code with lots of if statements

---

Error Names

errors have specific names, and Python shows them to us all the time.

```
>>> input_file = open("no_such_file.txt", 'r')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    input_file = open("no_such_file.txt", 'r')
IOError: [Errno 2] No such file or directory: 'no_such_file.txt'
>>> my_int = int('a string')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    my_int = int('a string')
ValueError: invalid literal for int() with base 10: 'a string'
>>>
```

recreate an error to find the correct name; spelling counts!

A Kind of Non-local Control

basic idea:
- keep watch on a particular section of code
- if we get an exception, raise/throw that exception (let it be known)
- look for a catcher that can handle that kind of exception
- if found, handle it; otherwise let Python handle it (which usually halts the program)

Doing Better with Input

- in general, we have assumed that the input we receive is correct (from a file, from the user)
  - this is almost never true
- there is always the chance that the input could be wrong
- our programs should be able to handle such errors

Worse Yet, Input is Evil

- "Writing Secure Code," by Howard and LeBlanc
  - "All input is evil until proven otherwise"
- most security holes in programs are based on assumptions programmers make about input
- secure programs protect themselves from evil input

Rule 7

All input is evil, until proven otherwise

General Form, Version 1

```
try:
  suite
except a_particular_error:
  suite
```

try Suite

- the `try` suite contains code that we want to monitor for errors during its execution
- if an error occurs anywhere in that `try` suite, Python looks for a handler that can deal with the error
- if no special handler exists, Python handles it, meaning the program halts and with an error message as we have seen so many times ☹

except Suite

- an `except` suite (or perhaps multiple `except` suites) is associated with a `try` suite
- each exception names a type of exception it is monitoring for
- if the error that occurs in the `try` suite matches the type of exception, then that `except` suite is activated

6/6/2019

try/except Group

- if no exception in the **try** suite, skip all the **try/except** to the next line of code
- if an error occurs in a **try** suite, look for the right exception
- if found, run that **except** suite and then skip past the **try/except** group to the next line of code
- if no exception handling found, give the error to Python

"The Practice of Computing Using Python, 3rd Edition", Punch & Enbody, Copyright © 2017 Pearson Education, Inc.

Code Listing 6.7
Find a line in a file

"The Practice of Computing Using Python, 3rd Edition", Punch & Enbody, Copyright © 2017 Pearson Education, Inc.

```
# read a particular line from a file. User provides both the line
# number and the file name

file_str = input( "Open what file:" )
find_line_str = input( "Which line (integer):" )

try:
    input_file = open(file_str)          # potential user error
    find_line_int = int(find_line_str)   # potential user error
    line_count_int = 1
    for line_str in input_file:
        if line_count_int == find_line_int:
            print("Line {} of file {} is {}".format(find_line_int, file_str, \
                  line_str))
            break
        line_count_int += 1
    else:
        # get here if line sought doesn't exist
        print("Line {} of file {} not found".format(find_line_int, file_str))
    input_file.close()

except FileNotFoundError:
    print("The file", file_str ,"doesn't exist.")

except ValueError:
    print("Line",find_line_str," Isn't a legal line number.")

print("End of the program")
```

"The Practice of Computing Using Python, 3rd Edition", Punch & Enbody, Copyright © 2017 Pearson Education, Inc.

Counting Poker Hands

Rank	Name	Description
9	Royal flush	(Ace, king, queen, jack, ten) + flush
8	Straight flush	Straight + flush
7	Four of a kind	Four equal ranks within five cards
6	Full house	Pair + different rank three of a kind
5	Flush	Five cards with the same suit
4	Straight	Five cards, sequentially ranked with no gaps
3	Three of a kind	Three equal ranks within five cards
2	Two pairs	Two pairs of equal ranks within five cards
1	One pair	One pair of equal ranks within five cards
0	Nothing in hand	

```
3, 10, 1, 7, 2, 12, 4, 2, 2, 1, 0
4, 9, 4, 12, 4, 13, 2, 6, 3, 4, 0
3, 2, 2, 2, 3, 12, 3, 1, 4, 2, 3
4, 11, 2, 8, 1, 13, 4, 7, 1, 7, 1
4, 8, 3, 8, 1, 3, 1, 4, 3, 7, 1
2, 7, 2, 5, 3, 10, 4, 13, 3, 7, 1
1, 4, 3, 4, 3, 5, 4, 10, 2, 10, 2
```

C1-suit, C1-rank, C2-suit, C2-rank, C3-suit, C3-rank, C4-suit, C4-rank, C5-suit, C5-rank, Hand Rank

"The Practice of Computing Using Python, 3rd Edition", Punch & Enbody, Copyright © 2017 Pearson Education, Inc.

Reminder, rules so far

1. Think before you program!
2. A program is a human-readable essay on problem solving that executes on a computer.
3. The best way to improve your programming and problem solving skills is to practice!
4. A foolish consistency is the hobgoblin of little minds
5. Test your code, often and thoroughly
6. If it was hard to write, it is probably hard to read. Add a comment.
7. All input is evil, unless proven otherwise.

"The Practice of Computing Using Python, 3rd Edition", Punch & Enbody, Copyright © 2017 Pearson Education, Inc.