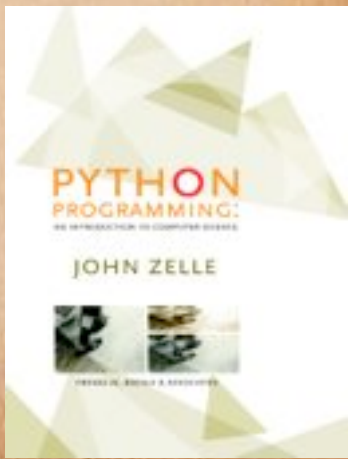
Python Programming: An Introduction to Computer Science

Chapter 4 (End of Chapter)
File IO



Coming up: File Processing

File Processing

- The process of *opening* a file involves associating a file on disk with a variable.
- We can manipulate the file by manipulating this variable.
  - Read from the file
  - Write to the file

File Processing

- When done with the file, it needs to be *closed*. Closing the file causes any outstanding operations and other bookkeeping for the file to be completed.
- In some cases, not properly closing a file could result in data loss.

File Processing Sequence

1. Open the file
2. Read from the file
3. Close the file

File Processing

- Working with text files in Python
  - Associate a file with a variable using the open function
    <filevar> = open(<name>, <mode>)
  - Name is a string with the actual file name on the disk. The mode is either ‘r’ or ‘w’ depending on whether we are reading or writing the file. “a” for appending to an existing file. (“a” will also create a non-existent file)
  - Infile = open(“numbers.dat”, “r”)

File Processing

- <filevar>.read() – returns the entire remaining contents of the file as a single (possibly large, multi-line) string
- <filevar>.readline() – returns the next line of the file. This is all text up to *and including* the next newline character
- <filevar>.readlines() – returns a list of the remaining lines in the file. Each list item is a single line including the newline characters.

File Processing: read

```
# printfile.py
#     Prints a file to the screen.

def main():
    fname = raw_input("Enter filename: ")
    infile = open(fname,'r')
    data = infile.read()
    print data

main()
```

- First, prompt the user for a file name
- Open the file for reading through the variable infile
- The file is read as one string and stored in the variable data

File Processing : readline

- readline can be used to read the next line from a file, including the trailing newline character

```
infile = open(someFile, ‘r’)
for i in range(5):
        line = infile.readline() # Read a single line
        print line[:-1] # Slice off the newline
```

- This reads the first 5 lines of a file
- Slicing is used to strip out the newline characters at the ends of the lines

File Processing: readlines

- Another way to loop through the contents of a file is to read it in with readlines and then loop through the resulting list.

```
infile = open(someFile, 'r')
for line in infile.readlines():
    # Line processing here
infile.close()
```

File Processing: easiest way!

- Python treats the file itself as a sequence of lines!
- infile = open(someFile), ‘r’)
  for line in infile:
      # process the line here
  infile.close()

File Processing: writing

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file's contents. If the named file does not exist, a new one is created.
- Outfile = open("mydata.out", 'w')
- <filevar>.write(<string>)

Warning: If you open an existing file for writing you DELETE EXISTING CONTENT of the file!!

File Processing : Writing

```
outfile = open("example.out", 'w')
count = 1
outfile.write("This is the first line\n")
count = count + 1
outfile.write("This is line number %d" % (count))
outfile.close()
```

- If you want to output something that is not a string you need to convert it first. Using the string formatting operators are an easy way to do this.

```
This is the first line
This is line number 2
```

Example Program: Batch Usernames

- *Batch* mode processing is where program input and output are done through files (the program is not designed to be interactive)
- Let's create usernames for a computer system where the first and last names come from an input file.

Example Program: Batch Usernames

```
# userfile.py
#    Program to create a file of usernames in batch mode.

import string

def main():
    print "This program creates a file of usernames from a"
    print "file of names."

    # get the file names
    infileName = raw_input("What file are the names in? ")
    outfileName = raw_input("What file should the usernames go in? ")

    # open the files
    infile = open(infileName, 'r')
    outfile = open(outfileName, 'w')
```

Example Program: Batch Usernames

```
# process each line of the input file
  for line in infile:
      # get the first and last names from line
      last, first = string.split(line, “,”) # Split the names on comma
      # create a username
      uname = string.lower(first[0]+last[:7])
      # write it to the output file
      outfile.write(uname+'\n')

  # close both files
  infile.close()
  outfile.close()

  print "Usernames have been written to", outfileName
```

Example Program: Batch Usernames

- Things to note:
  - It's not unusual for programs to have multiple files open for reading and writing at the same time.
  - The lower function is used to convert the names into all lower case, in the event the names are mixed upper and lower case.
  - We need to concatenate '\n' to our output to the file, otherwise the user names would be all run together on one line.

Coming Attraction: Objects

- Have you noticed the dot notation with the file variable? *infile.read()*
- This is different than other functions that act on a variable, like abs(x), not x.abs().
- In Python, files are *objects*, meaning that the data and operations are combined. The operations, called *methods*, are invoked using this dot notation.
- Strings and lists are also objects. More on this later!

More info?

- Always more info in the book…
- On the web:

- http://docs.python.org/tut/node9.html