Programming in Python VI: Working with Files

Computer Science 105
Boston University
Spring 2014

David G. Sullivan, Ph.D.

Escape Sequences

- Recall: we can surround strings by either single or double quotes.
  - doing so allows us to embed quotes within a string

```
'Homer said, "Doh!"'
```

- We can also embed a double or single quote by preceding it with a `\` character.

```
"Homer said, \"Doh!\""
```

- `\"` is known as an *escape sequence*.
- The `\` tells the compiler to interpret the following character differently than it ordinarily would.
- Other examples:
  - `\n` a newline character (go to the next line)
  - `\t` a tab
  - `\\` a backslash!

Text Files

- A text file can be thought of as a multi-line string.
  - example: the following four-line text file

```
# simple.py

print(2 + 3)
print(10 – 7)
```

is equivalent to the following string:

```
"# simple.py\n\nprint(2 + 3)\nprint(10 – 7)\n"
```

Opening a Text File

- Before we can read from or write to a text file, we need to *open* a connection to the file.
- Doing so creates an object known as a file handle.
  - we use the file handle to perform operations on the file
- Syntax:

```
<file-handle> = open(<filename>, <mode>)
```

where <file-handle> is a variable for the file handle
<filename> is a string
<mode> is:

- `'r'` if we want to read from the file
- `'w'` if we want to write to the file, (erasing any existing contents)
- `'a'` if we want to append to the end of the file

Specifying Filenames

- When specifying the name of a file, we'll just give the name of the file itself.
  - example: `"myData.txt"`
  - we won't specify the directory
- Python will open/create the file in the same directory in which the module file is stored.
  - the rules are a bit more complicated when you're using the interpreter from the command prompt

Closing a File

- Here's an example of opening a file for writing:

```
outfile = open("example.txt", 'w')
```

- When we're done reading from or writing to a file, we need to close its handle:

```
outfile.close()
```

- **Important:** Text that you write to file may not make it to disk until you close the file handle!

Writing to a File

- When you open a file for writing:
  - if the file doesn't already exist, it will be created
  - if the file does exist and you specify the `'w'` mode, the current contents will be erased!
  - if the file does exist and you specify the `'a'` mode, the text you write will be appended to the end of the file
- To write values to a file, we can use the `print()` method as usual, but with an extra parameter for the file:

  `print(..., file=<file-handle>)`

- Example:

```
outfile = open("foo.txt", 'w')
print("I love Python!", file=outfile)
```

Example: Writing Database Results to a File

- Recall our program for getting all movies from a given year:

```
import sqlite3
filename = input("name of database file: ")
db = sqlite3.connect(filename)
cursor = db.cursor()

searchYear = input("year to search for? ")
command = '''SELECT name, rating
             FROM Movie WHERE year = ?;'''
cursor.execute(command, [searchYear])

for tuple in cursor:
    print(tuple[0], tuple[1])

db.commit()
db.close()
```

- Let's modify it so that it writes the results to a file.

Reading from a File

- When you open a file for reading, the file must already exist, or you'll get an error:

```
>>> infile = open("noexist.txt", 'r')
…IOError: [Errno 2] No such file or directory
```

- To read one line at line at a time, we can use the `readline()` function.
- Syntax:

```
<variable> = <file-handle>.readline()
```

example: `line = infile.readline()`

- This function returns a string containing the next line in the file – up to and including the next newline character ('\n').

Reading from a File (cont.)

- Example: assume that we have our earlier four-line text file:

```
# simple.py

print(2 + 3)
print(10 – 7)
```

  - here's one possible set of operations on that file:

```
>>> infile = open("simple.py", 'r')
>>> line = infile.readline()
>>> line
'# simple.py\n'
>>> infile.readline()
'\n'
>>> line = infile.readline()
>>> line
'print(2 + 3)\n'
```

Closing and Reopening a File

- If we've been reading from a file and want to start over again from the beginning of the file, we need to close the file and reopen it again.
  - example:

```
>> infile = open("simple.py", 'r')
>> infile.readline()
'# simple.py\n'
>> infile.readline()
'\n'
>> infile.close()
>> infile = open("simple.py", 'r')
>> infile.readline()
'# simple.py\n'
```

Processing a File Using a `for` Loop

- We often want to read and process each line in a file.
- Because we don't usually know how many lines there are in the file, we use a `for` loop.
- Syntax:

```
for line in <file-handle>:
    # code to process line goes here
```

  - reads one line at a time and assigns it to `line`

Example of Processing a File

- Let's say that we want a program to print a text file to the screen, omitting all blank lines.
- Here's one possible implementation:

```
filename = input("name of file: ")
infile = open(filename, 'r')

for line in infile:
    if line != "\n":
        print(line[:-1])

infile.close()
```

- Why do we need to use slicing? (`line[:-1]`)
- How could we change it so that it omits full-line comments instead (i.e., lines that begin with a #)?

Extracting Relevant Data from a File

- High-school and college track teams often participate in meets involving a large number of schools.
- Assume that the results of a meet are summarized in a comma-delimited text file that looks like this:

```
Mike Mercury,Boston University,mile,4:50:00
Steve Slug,Boston College,mile,7:30:00
Len Lightning,Boston University,half-mile,2:15:00
Tom Turtle,UMass,half-mile,4:00:00
```

- Let's write a program that reads in a results file and extracts just the results for a particular school – with the name of the school omitted from the records.
  - write the results to a file

Extracting Relevant Data from a File