Chapter 9

More with Files

In chapter 6, we saw how to print to a file instead of to the screen. In this chapter, we will see how to read information from existing files. Then we will write programs to process data from files, and to edit files.

Reading from files

We shall consider the opening paragraph of Kafka's "Metamorphosis".

```
One morning, when Gregor Samsa woke from troubled dreams, he found
himself transformed in his bed into a horrible vermin.  He lay on
his armour-like back, and if he lifted his head a little he could
see his brown belly, slightly domed and divided by arches into stiff
sections.  The bedding was hardly able to cover it and seemed ready
to slide off any moment.  His many legs, pitifully thin compared
with the size of the rest of him, waved about helplessly as he
looked.
```

There are newline characters at the end of each line, save for the last. You can cut and paste or type this into a text file to try these examples out. Here, it is saved as `gregor.txt`. Now, we can read the whole contents of the file into a string using `'r'` for reading mode:

```
Python
>>> f = open('gregor.txt', 'r')
>>> f.read()
'One morning, when Gregor Samsa woke from troubled dreams, he found\nhimself tran
sformed in his bed into a horrible vermin.  He lay on\nhis armour-like back, and
if he lifted his head a little he could\nsee his brown belly, slightly domed and
divided by arches into stiff\nsections.  The bedding was hardly able to cover it
and seemed ready\nto slide off any moment.  His many legs, pitifully thin compare
d\nwith the size of the rest of him, waved about helplessly as he\nlooked.'
>>> f.read()
''
```

This single string contains the `\n` newline characters, of course. If we call `f.read()` again, the result is the empty string. This is because there is nothing else left to read – the contents of the file has already been read and we are at the end of the file.

Three ways to iterate over lines

Instead of reading the whole file as one big string, we may read the lines in turn, by repeated use of the `readline` method:

```
Python
>>> f = open('gregor.txt')
>>> f.readline()
'One morning, when Gregor Samsa woke from troubled dreams, he found\n'
>>> f.readline()
'himself transformed in his bed into a horrible vermin.  He lay on\n'
>>> f.readline()
'his armour-like back, and if he lifted his head a little he could\n'
>>> f.readline()
'see his brown belly, slightly domed and divided by arches into stiff\n'
>>> f.readline()
'sections.  The bedding was hardly able to cover it and seemed ready\n'
>>> f.readline()
'to slide off any moment.  His many legs, pitifully thin compared\n'
>>> f.readline()
'with the size of the rest of him, waved about helplessly as he\n'
>>> f.readline()
'looked.'
>>> f.readline()
''
```

Notice that we omit the `'r'` argument to the `open` function – it is the default. Again, we know that there is no more to read when the result is the empty string. We can, alternatively, iterate directly over the contents of the file with a **`for`** loop:

```
Python
>>> f = open('gregor.txt')
>>> for line in f:
...   print(line, end='')
...
One morning, when Gregor Samsa woke from troubled dreams, he found
himself transformed in his bed into a horrible vermin.  He lay on
his armour-like back, and if he lifted his head a little he could
see his brown belly, slightly domed and divided by arches into stiff
sections.  The bedding was hardly able to cover it and seemed ready
to slide off any moment.  His many legs, pitifully thin compared
with the size of the rest of him, waved about helplessly as he
looked.
```

Finally, we can use the `list` function to return a list of all the lines in the file in one go:

```
Python
>>> f = open('gregor.txt')
>>> list(f)
['One morning, when Gregor Samsa woke from troubled dreams, he found\n', 'himself
 transformed in his bed into a horrible vermin.  He lay on\n', 'his armour-like b
ack, and if he lifted his head a little he could\n', 'see his brown belly, slight
ly domed and divided by arches into stiff\n', 'sections.  The bedding was hardly
able to cover it and seemed ready\n', 'to slide off any moment.  His many legs, p
itifully thin compared\n', 'with the size of the rest of him, waved about helples
sly as he\n', 'looked.\n']
```

Example: reversing lines

We can write a program to read all the lines from a file, and write them in reverse order to another file:

```
Python
>>> f = open('gregor.txt')
>>> f_out = open('output.txt', 'w')
>>> for x in reversed(list(f)):
...     print(x, end='', file=f_out)
...
>>> f.close()
>>> f_out.close()
```

Here is the contents of the output file:

```
looked.
with the size of the rest of him, waved about helplessly as he
to slide off any moment.  His many legs, pitifully thin compared
sections.  The bedding was hardly able to cover it and seemed ready
see his brown belly, slightly domed and divided by arches into stiff
his armour-like back, and if he lifted his head a little he could
himself transformed in his bed into a horrible vermin.  He lay on
One morning, when Gregor Samsa woke from troubled dreams, he found
```

We can use an extended version of the **`with`** … **`as`** structure we have already seen to prevent mistakes with matching up the opening and closing of files. Here is the same program in this simpler, safer, form:

```
Python
>>> with open('gregor.txt') as f, open('output.txt', 'w') as f_out:
...     for x in reversed(list(f)):
...         print(x, end='', file=f_out)
```

Files and exceptions

Not only does the **with** ... **as** construct prevent double-closing of a file, but also prevents any attempt to read from a file which has already been closed:

```
Python
>>> f = open('gregor.txt')
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

There are still, however, some exceptions we may need to handle, even when using **with** ... **as** – for example, a missing file:

```
Python
>>> open('not_there.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'not_there.txt'
```

Example: text statistics

Consider this program to return the number of lines, characters (letters or other symbols), words, and sentences in a given file:

```
#Text statistics: lines, characters, word, sentences.
def is_full_stop(s):
    return s == '.'

def stats_from_file(f):
    lines = 0
    characters = 0
    words = 0
    sentences = 0
    for line in f:
        lines += 1
        characters += len(line)
        words += len(line.split())
        sentences += len(list(filter(is_full_stop, line)))
    return (lines, characters, words, sentences)

def stats_from_filename(filename):
    with open(filename) as f:
        return stats_from_file(f)

gregor_stats = stats_from_filename('gregor.txt')
```

Notice we can use `filter` directly on `line` without turning it into a list. Here is the result:

```
Python
>>> gregor_stats
(8, 472, 85, 4)
```

That is to say, 8 lines, 472 characters, 85 words, and 4 sentences. In the questions, you will be asked to extend this program to collect more statistics.

Common problems

In addition to situations which can lead to file-related exceptions, there are two more common issues which can occur when processing files. If we open a file which already exists, with the intention of writing to it, but we forget to open it in `'w'` mode, an exception occurs:

```
Python
>>> with open('exists.txt') as f:
...     print('output', file=f)
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
io.UnsupportedOperation: not writable
```

When printing lines from a file which we have read using, for example, `readlines`, it is important to remember that they will always have a `\n` newline at the end already. If we print them just using plain `print` we will see double spacing:

```
Python
>>> f = open('gregor.txt')
>>> for line in f:
...   print(line)
...
One morning, when Gregor Samsa woke from troubled dreams, he found

himself transformed in his bed into a horrible vermin.  He lay on

his armour-like back, and if he lifted his head a little he could

see his brown belly, slightly domed and divided by arches into stiff

sections.  The bedding was hardly able to cover it and seemed ready

to slide off any moment.  His many legs, pitifully thin compared

with the size of the rest of him, waved about helplessly as he

looked.
```

Summary

We now know how to read from files as well as write to them. This means we can write file-processing programs, which read from one file, process the data in some way, and write to another. This is a significant class of useful programs. We made our programs cleaner and less error-prone by extending our use of the **`with`** … **`as`** construct. We learned how to iterate over the lines in the file, and so over the characters in each line, building our file statistics program. In some of the questions, you will be asked to extend this file statistics program in various ways.

In the next chapter, we fill in another gap in our knowledge: the real numbers – that is to say the ones which are not whole numbers.

Questions

1. We wrote a program to print out the contents of a file line-by-line:

   ```
   Python
   >>> f = open('gregor.txt')
   >>> for line in f:
   ...     print(line, end='')
   ```

   Rewrite this program using the **with** ... **as** construct.

2. Give a function to write a dictionary with integer keys and string values to a given file. For example, the dictionary `{1: 'oak', 2: 'ash', 3: 'lime'}` should produce the file:

   ```
   1
   oak
   2
   ash
   3
   lime
   ```

3. Now write a function to read such a dictionary back from file. Make sure to handle exceptions arising from incorrect data. There is a built-in method `strip` which removes spaces and newlines from either end of a string which may prove useful.

4. When we write to a file which already exists, its contents are overwritten. The file mode `'a'` allows information to be appended to a file instead. Use this to write a function which concatenates two files, writing the result to a third.

5. Write a function which reads a file containing multiple numbers, separated by spaces, on multiple lines, and calculates their total.

6. Write a function `copy_file` which, given two file names, reads the contents of the first, and writes it to the second.

7. Extend our text statistics to print a histogram of the frequencies of each letter in the file. You might remember we wrote a similar histogram program in the questions to chapter 4.

8. Extend it again to print a histogram of frequencies of words. How might punctuation and capital letters be dealt with? Hints:

   - There is a built-in method `strip` on strings which can be given an argument, a string containing the characters to be stripped.
   - The string `string.punctuation` following **`import`** `string` contains common punctuation characters.
   - The `lower` method on a string method converts it to lowercase.

9. Write a function to search for a given word in a given file, listing the line numbers and lines at which it appears. Use our lessons from the previous question to deal with punctuation.

10. Write a function `top` which prints the first five lines from a file, waiting for the user to press Enter for another five, and so on.