1 File objects

Information stored in files can be accessed by a python program. To get access to the contents of a file, you need to open the file in your program. When you are done using a file, you close it.

Python has a built-in function `open` to open a file. The function call is `open(filename, mode)`, where `mode` is `'r'` (to open for reading), `'w'` (to open for writing), or `'a'` (to open for appending to what is already in the file). The `mode` parameter is *optional*: if omitted, it defaults to `'r'`.

This opens a file called `RoadNotTaken.txt` for reading:

```
f = open('RoadNotTaken.txt', 'r')
```

Important: the file *must* be in the same folder as your program. If you want to open a file in a different folder, the first argument to `open` must be the *path* to the file. The full path for the above file on my computer is `'/Users/mhay/Desktop/cosc101/RoadNotTaken.txt'`. Also, you must provide the full path if you want to open files in IDLE's interactive mode (i.e., in the shell).

The function `open` returns a file object. The file object keeps track of its *position* in the file. To close this file, you write `f.close()`.

2 Reading from files

There are four standard approaches to read from a file. Some use these methods:

- `readline()`: reads and returns the next line from the file, including the newline character (if it exists). Returns the empty string if there are no more lines in the file.
- `readlines()`: read and return all lines in a file in a list. The lines include the newline character. *Not recommended for large files.*
- `read()`: read the whole file as a single string. *Not recommended for large files.*

Approach #1: the `while` loop approach. This approach is useful when we only want to process *some* of the file. For example, this program prints the first stanza of a poem.

```
frost_file = open('RoadNotTaken.txt', 'r')
title = frost_file.readline()       # skip title
blank_line = frost_file.readline()  # skip blank line
line = frost_file.readline()        # read first line of stanza
while line != '\n':                 # blank line indicates end of stanza
    print line,                     # comma because line has its own newline
    line = frost_file.readline()
frost_file.close()
```

Approach #2: the `for` loop approach. This approach is preferred when we want to process the *entire* file. For example, this program counts the number of times "road" appears in the poem.

for loop approach

```
frost_file = open('RoadNotTaken.txt')
count = 0
for line in frost_file:
    if 'road' in line:
        count += 1
frost_file.close()
print "The poem contains", count,
print "occurrences of 'road'."
```

equivalent while loop

```
frost_file = open('RoadNotTaken.txt')
count = 0
line = frost_file.readline()
while line != '':
    if 'road' in line:
        count += 1
    line = frost_file.readline()
frost_file.close()
print "The poem contains", count,
print "occurrences of 'road'."
```

Important detail: After the `for` loop finishes, the file object is at the *end* of the file. To read the file again, you will need to close it and re-open it.

Approach #3: use the `read` method to read the *entire* file into memory and use it as a single string.

Approach #4: use the `readlines` method to examine each line of a file by index.

```
frost_file = open('RoadNotTaken.txt', 'r')
contents_list = frost_file.readlines()
frost_file.close()
print "The last line is", contents_list[-1]
```

3 Exercises

Some solutions are presented in class and also included in the moodle version of this handout. The files used in these examples are available on moodle.

1. Open `'RoadNotTaken.txt'` and print just the first stanza of poem. In other words, skip over the title and the blank line, then print lines until you encounter the next blank line. Use a while loop for this exercise.

Solution:

```
frost_file = open('RoadNotTaken.txt', 'r')
frost_file.readline() # skip title
frost_file.readline() # skip blank line

# read first stanza
line = frost_file.readline()
while line != '\n':
    print line.strip()
    line = frost_file.readline()
frost_file.close()
```

2. Modify the program to print the *second* stanza. Hint: add a second while loop.

Solution:

```
frost_file = open('RoadNotTaken.txt', 'r')
frost_file.readline() # skip title
frost_file.readline() # skip blank line

# read first stanza
line = frost_file.readline()
while line != '\n':
    line = frost_file.readline()
line = frost_file.readline()
while line != '\n':
    print line.strip()
    line = frost_file.readline()
frost_file.close()
```

3. Write a program that reads `'carroll_acrostic.txt'`, which contains an acrostic poem, and prints out the acrostic – i.e., the phrase formed by just looking at the first letter of each line. If the poem has multiple stanzas, do *not* include the blank lines. Print the acrostic as a single string without spaces. Use a for loop.

Solution:

```
f = open('carroll_acrostic.txt')
phrase = ''
for line in f:
    if line.strip() != '':
        phrase += line[0]
f.close()
print phrase
```

Some material adapted from Gries and Campbell.