Reading From and Writing To Files

The following example programs illustrate some issues related to reading from and writing to files.

Working with a partner: Download `example1.py`, `example2.py`, `example3.py`, `example4.py` and `exParrot.txt` from the Artifacts section of this week’s website to a new directory for this week’s class. Bring up Spyder. Set the directory containing the downloaded files as the working directory. Run the following experiments.

1. Example #1:
   a. Open `example1.py` in Spyder. Before running it, be sure the directory does not contain a file called `out1.txt` (delete the file if there is one) and discuss with a partner what affect running the program will produce. Run the program once and check if your prediction was correct. What can you conclude from this experiment?

   b. Delete the comment character (#) in front of the last line of the program and discuss with a partner what affect running the program a second time will produce. Run the program a second time and check if your prediction was correct. What can you conclude from this experiment?

   c. Modify `example1.py` so that it always writes the data to the file. Delete the file `out1.txt` and rerun the program to check that the program now behaves as expected on the first run.

2. Example #2:
   a. Open `example2.py` in Spyder. Before running it, be sure the directory does not contain a file called `out2.txt` (delete the file if there is one) and discuss with a partner what affect running the program will produce. Run the program once and check if your prediction was correct. Compare the results you get for experiment #2 with those of others at your table. What can you conclude from this experiment?

   b. Add an instruction to `example2.py` to close the file after the loop is executed. Rerun the program. Check that it produces the expected results.

3. Example #3:
   a. Open `example3.py` in Spyder and run it to see the error it produces.

   b. Consulting the files cheat sheet, as needed, modify this program so that it uses the `write` method to create `out3.txt` with the exact same contents as the `out2.txt` file produced in 2b.
4. Example #4:
   a. Open `example4.py` in Spyder. Before running it, discuss with your partner what the file produced by running the program will contain. Run the program once and see if your prediction was correct. What can you conclude from this experiment about the input file? What is causing a blank line to be produced between each numbered line?

   b. Modify `example4.py` so that executing it produces a file containing just the numbered lines, with no blank lines between them. Thus:

   ![Image of example4.txt with lines numbered]

   c. Further modify `example4.py` so that it numbers only the lines that have text in them, and it preserves blank lines. Thus:

   ![Image of example4.txt with lines numbered and blank lines preserved]
d. Add a function `leftJustify(orig, size)` that, for a string `orig` and positive int `size`, returns the string produced by inserting a new-line character ('\n') into `orig` after every `size` characters. Test that your function works correctly, then modify `example4.py` to use the results returned by calling `leftJustify` with `size` equal to 80 to produce a file formatted as shown below.

```python
10 Mr. Praline:
    All right then, if he's restin', I'll wake him up! (shouting at the cage) 'Ello,
    'Mister Polly Parrot! I've got a lovely fresh cuttle fish for you if you show...
    (owner hits the cage)
11 Owner:
    There, he moved!
12 Mr. Praline:
    No, he didn't, that was you hitting the cage!
13 Owner:
    I never!!
14 Mr. Praline:
    Yes, you did!
15 Owner:
    I never, never did anything...
16 Mr. Praline:
    (yelling and hitting the cage repeatedly) 'ELLO POLLY!!!! Testing! Testing! Tes-
    ting! Testing! This is your nine o'clock alarm call!
    (Takes parrot out of the cage and thumps its head on the counter. Throws it up i
    n the air and watches it plummet to the floor.)
17 Mr. Praline:
    Now that's what I call a dead parrot.
```
e. Modify the `leftJustify` function so that it does not break words, but inserts the newline character so as to leave a ragged right margin while left justifying the text in a column of width `size`. Thus, running `example4.py` with `size` equal to 80 in the call to `leftJustify` produces: