CISC106 Fall 2011 Lab07

- This lab an all subsequent labs will be due Sunday at 11:55 PM EDT on Sakai.
- The preparation problems below are to develop your understanding without creating extra work for you or the TA; these problems will not be graded. Be sure to read and understand them - they will help with the problems you must submit for grading
- Review the code examples from your notes in class.
- You may work with one or two other people on your lab (max size is three!). These people must be in your same lab section. If you do, **one** of you should be designated to submit the assignment on Sakai. **All of your names** should appear on code that you develop together[1].
- Whom do you think deducts more points: a happy TA, or a frustrated TA? Make your work easy to read! It isn't just good software engineering, it is good for your grade!
- EVERY python program/function must include header, doc string that contains a human-readable desciption of what the function does, and must be followed by a good series of tests, as discussed in class. Always test boundaries. Do not test erroneous input (e.g. a factorial function does not need to correctly handle strings).
- EVERY .py file must have a comment line at the very top containing your name(s), lab section, and a brief description of what the file is.
- Write the tests first! Real software engineers do this for very good reasons - so should you! (This of course applies as well to parts one and two, but the tests are already given to you for those.)

Preparation (do not submit for grading)

1. Copy the following bit of code into the interpreter:

```
try:
    raise RuntimeError
    print 'Does this message show up at all?'
except RuntimeError:
    print 'This message had better show up.'
```

   Carefully examine what happens.

2. type the following into the interpreter and examine what gets printed out:

```
>>> x = 'a strings'
>>> x[0]
>>> x[:-1]
```

[1] If you would like to work with someone but don't know whom, your TA may be able to help connect you to other students looking for lab partners.

```
>>> x = 'some other string'
>>> x[5:10]
>>> x = 'a-bunch-of-words'
>>> x.split('-')
```

Programs (to be graded)

1. Download the files `ff7.pyc`, `lab07_tests.py`, `save00.ff7` and `save05.ff7` from the course website. Now create a `lab07.py` and make sure you `import ff7` in it. Make a class `FF7Save` who's constructor takes a string and builds $self$ from that string. An FF7Save should contain attributes representing the following: Name, current level, current and max HP and current and max MP of the lead character, Names of the three characters in the party[2], amount of money held by the party and their current location. See the test in `lab07_tests.py` for the format of the input string.

2. Now implement the special function *__unicode__* for FF7Save. The `__unicode__` function takes an object and returns a (unicode) string representation of that object. It will be used when an FF7Save is cast as a string. The returned string should be in *Comma-separated values* (CSV) format. Again, see the test for examples of what a CSV row looks like.[3] When you are done with this part, you should be able to run the test and watch it pass.

3. Now you should implement a function `read_saves` which takes a path $fname$ and reads in all of the saves from $fname$ into a list.[4] N.B. that you'll need to open the save file as a *binary file*. You can do this by giving the call to open the appropriate flag, *e.g.* `fin = open(fname, 'rb')`. The `b` in the second parameter of course stands for binary. An ff7 save file contains at most 15 saves, but may contain less. If you try to read past the last save, it will raise `IOError`. You should make sure your function will successfully read in all saves regardless of how many are in the file. To read a save, you should call `ff7.read_save_string` and hand it the open binary file. In your test for this function, you may simply read save00.ff7 and save05.ff7 into lists and make sure the lists then contain the correct number of saves. There are 11 saves in save00.ff7 and 15 in save05.ff7.

4. Now you should implement a function `write_saves_to_csv` which takes a path $fname$ and a list of saves $saves$. It will write a CSV file containing all of the saves in $saves$ to the file at $fname$. N.B. that you should write a header to your CSV file before writing any of the saves. The header is just a comma-separated list of column names, *e.g.* `'Name,Level,Location'` would be a three column header where the column names are `Name`, `Level` and `Location`, respectively. You should make sure to put your column titles in the correct order, which you know based on how you implemented `__unicode__` on your `FF7Save` class.[5] You should test this function in a manner similar to how you tested `write_list` in lab 05. For the sake of your sanity, you may choose to build a list of the three examples in `text_make_FF7save` and write that list to a file for your test.

[2] if there are less than three characters in the party, you should keep track of this fact as well.

[3] for more information on Comma-separated values (or CSV) files, see `http://en.wikipedia.org/wiki/Comma-separated_values`

[4] You can get the string representation of a save by calling `ff7.read_save_string` and passing it the open save file.

[5] If you have Excel, OpenOffice.org or some other spreadsheet program, you can open your CSV file in it to view it like a spreadsheet.

5. Finally, you should download `saveviewer.py` and `portraits.py` from the course website. Run saveviewer.py and see what options it presents you with. You'll need to find and modify the functions called by the *Read Saves* and *Write CSV* buttons so that they use your `read_saves` and `write_saves_to_csv` functions, respectively. Once you've done that, you should look at the `SaveDisplayWidget` class and modify its cunstructor so that it correctly uses the `FF7Save` given to it to build its display.[6]

Make sure you eventually call `close()` on any file you `open()`. You should submit your lab07.py, lab07_tests.py, saveviewer.py, any CSV files you created, and any other docs required by your TA on Sakai.

[6] What's happening here is known as the *Model/View pattern*. `FF7Save` is the *model* and `SaveDisplayWidget` is its *view*.