PIC 16, Winter 2018 – Preparation 4M

Assigned 1/26/2018. To be completed by class 1/29/2018.

Intended Learning Outcomes

By the end of this preparatory assignment, students should be able to:

- convert numerics and objects to strings,
- get text and command input from the command line,
- read and write text from/to files, and
- read and write data from/to csv files using the `csv` module.

Tasks

- ☐ Read 7.1. We'll talk about it more in class.
- ☐ Read 7.2 and 7.2.1. You've already practiced reading from a file, now note how to open a file for writing and write to the file. Also, the `with` keyword is new. We won't need the `tell` or `seek` methods in this class.
- ☐ Write `hello world` to a new file `helloworld.txt`. Don't create the file in a text editor first; let Python create the file for you. Make it such that running the program multiple times should add *additional* lines `hello world` to the file rather than overwriting it. Please open the file in a text editor to make sure your program is working correctly.
- ☐ You might have looked up that the `r+` mode is for "reading and writing", and you might be wondering how you modify the content of files in place. These sound like good ideas until you think about the details. Using the `r+` mode and modifying files in place require thinking about the file byte by byte. It's not very user friendly, and you can only *replace* bytes (not insert information between existing bytes), so it's not typically done unless efficiency is *really* important. If you want to check the contents of an existing file and modify the contents:
    - open the file for reading,
    - read the whole thing,
    - close the file,
    - modify the data you read, then
    - write the whole thing back.

  See here for a more advanced ideas. If the file is very large, you might want to open file A for reading and open a new file B for writing and then in a loop
    - read a small chunk from file A,
    - modify it, and
    - write the modified chunks to file B

  so that you don't have to load the entire file into memory at once. Then you can delete file A and rename file B. If you're curious how it `r+` can work, see here for a simple example, but again, we won't use `seek` or `tell` in this class.
- ☐ Read 7.2.2.
- ☐ Practice with the dictionary
  `x = {"Hello":1, "Goodbye": 2}`
  First save it as text using the `with` ... `as` syntax and read it back into a different variable, `y`. Does
  `print y["Hello"]`
  show 1? What kind of variable is `y`?
  Now dump it using the `json` module and load it back into `y`. Now does
  `print y["Hello"]`
  show 1? What kind of variable is `y`?

- ☐ Read “The *input* function” from http://www.tutorialspoint.com/python/python_files_io.htm
- ☐ Write a program that prompts the user to enter a math expression and evaluates the result. If the expression is invalid, the program should inform the user and quit. Example:

```
Enter math expression:
Expression: 1+1
Result:     2
Expression: 3*2
Result:     6
Expression: 4**2
Result:     16
Expression: q
Invalid Expression
```

- ☐ Read the *raw_input* function”
- ☐ Now, modify your program such that only a user input of `q` or `quit` (Case insensitive, WITHOUT QUOTES) causes the program to quit. If the input is invalid Python code, print `Invalid Expression` and continue. If the expression is valid Python code but not a typical math expression (what is common in typical math expressions?), notify the user but print the result anyway. (Hint: look up the `eval` method...).

```
Enter math expression, or 'q' to quit:
Expression: 1+1
Result:     2
Expression: sin(10)
Result:     -0.544021110889
Expression: sin(x)
Invalid Expression
Expression: "hello" + "world"
That wasn't a math expression!
Result:     helloworld
Expression: QuIt
```

  As you will see, the `raw_input` function is more flexible than `input`, and so in Python 3 the `input` command behaves like Python 2’s `raw_input`.
- ☐ Continue reading from Opening and Closing Files through File Positions.
- ☐ Read: http://www.pythonforbeginners.com/systems-programming/using-the-csv-module-in-python/ (or any other `csv` module tutorial you find that you prefer)
- ☐ Read in simpledata.csv into a single list (a list of lists, really) and print it with all the names (but not the header info) in ALL CAPS. Write this data (again with the names in all caps) to a new file `simpledata2.csv` with the same delimiter as the original file (open the file in a text editor to see...).
- ☐ Here’s a video on file reading and writing if you’re interested.