COM110: Lab 2

Numeric expressions, strings, and some file processing (chapters 3 and 5)

1) **The range() function**. We've already seen in class that `range(n)` can be thought of as a list of integers `[0,1,2,...,n-1]`. In our reading this and last week we learn that the `range()` function is actually more versatile that this! Look at each of the `range()` function calls from Discussion Exercise 3 on page 78 and see if you can predict the outcome. Test them in the Python interactive shell. If you still can't figure it out how `range()` works, keep playing around with calls to `range()` of your own, or refer to the reading starting on the bottom of page 70 to the middle of page 71. For this checkpoint you must show us the tests you ran and explain to us how the `range()` function works.

☺Get check 1☺

2) **Pizza: small or large?** At the Two Wives Pizza restaurant in downtown New London, you can get any of their pizzas in two sizes: 8 inches and 14 inches. For example, (last I checked) their four cheese pizza is $7 for the 8 inch size, and $11.50 for the 14 inch size. At first glance, you might think to yourself, "Okay, fine, $11.50 for 14 inches sounds like a *slightly* better deal than $7 for 8 inches." But remember, these inches refer to the pizza's *diameter* (length across the middle), while the *area* of the pizza in square inches is actually equal to πr^2, where r is the pizza's radius (half of the diameter). So for each extra inch of radius, the *area* (in *square inches*) of the pizza grows by a lot more than one!

Write a program that inputs prices for the small and the large pizzas at Two Wives, then calculates and compares the *price per square inch* for each. (You'll need to import the math module [Sec. 3.3 in your books] so that you can use the built-in Python `pi` function. That is, at the very top top of your module, use the command `import math`, then, whenever you want the (approximate) value for π, simply type `math.pi`.) .

NB: Your function should output information such as "The the price of square inch for the 8" pizza is $1.67/si where as for the 14" is $2.21/si"

☺Get check 2☺

3) **Practice with strings.**

   a. For discussion question 1 (on page 169) at the end of chapter 5, first see if you can predict how Python will evaluate each expression. (Refer to page 148, Table 5.2—or the internet!—if you can't remember what each string method does.) Then test your answers in the interactive Python interpreter. If any results don't make sense, be sure to ask!

   b. Complete discussion exercise 2 (on page 170) at the end of chapter 5. Hint: for part (e) use the .split() function, as explained on page 144.

☺Get check 3☺

4) **File Processing.** If you have a bunch of data or words, how can you do computations on it and mine it for information? One way is to put the data into a plain text file, and then have your python program "read" from it. So for this checkpoint, we are going to learn how to have our python programs read in the contents of a file.

- First, note that most files are not plain text files. For example, a very plain-looking Word document is not actually "plain" at all. It is littered with all sorts of characters that indicate to Microsoft Word - formatting, style, etc of the document.

- For our purposes, the input file being read from should be a text file (perhaps created in Notepad or TextEdit, but either way, make sure it is a *plain text* file, with a .txt extension, no formatting, fonts, etc), and it must be in the same folder where you have saved the python program you are writing. Save a copy of the file lab2poem.txt into the same folder that you are saving the current lab module in.

- Have your program "open" the file to read from it using the following line of code:

```
<input file variable name> = open(<file name>, "r")
```

  It creates a variable that refers to a "file object" type, opens the file, and specifies that we wish to ***read*** from that file (as opposed to *write* to the file).

- E.g., if we want "inputfile" to be the variable name for our input file, we would type

```
inputfile = open("lab2poem.txt", "r")
```

Don't worry if it appears as though nothing has happened. After this line of code executes, the file is "open" in the computer's memory for reading by the program you are writing (not open on the screen for reading by the human user).

- Once you `open` the file for reading, there are several options (different commands) for reading the text into our python program. In all the following cases there is a "pointer" (an imaginary cursor) that starts at the beginning of the file and then moves through the text with each command.

    a. **`read()`**. The following line of code

    `<variable> = inputfile.read()`

    will read all of the text in the file into one long string and store it in the variable (newline characters—remember "`\n`"?—are embedded in the string). It will also leave the "pointer" at the end of the file (nothing more can be read in). Try doing this and then printing the contents of the variable.

    b. Files must then be closed (after you finish reading or writing to them) by calling the `close()` function:

    `inputfile.close()`

    ☺Get check 4☺

    c. **`readline()`**. In a new Python module, again `open()` the file as done earlier. Then the following line of code

    `<variable> = inputfile.readline()`

    will read one line of the file (with the "`\n`" on the end) into a variable and put the pointer at the beginning of the next line. Try using this command to

read a couple lines of the file into a couple different variables. Print the variables out to make sure it's working as you expect.

d. **`readlines()`**. The following line of code

```
<variable> = inputfile.readlines()
```

creates a **list** of all the remaining lines of the file (with a "`\n`" on the end of each line in the list) and puts the list into a variable, leaving the pointer at the end of the file. Try it in the same module as the last one, again printing out the contents of the variable to make sure it's working. It should start reading the file from where you last left off, since you never closed and re-opened the file.

☺Get check 5☺

e. Note that once the file is "read" all the way to the end using one (or a combination) of these commands, trying to `read` it further will produce empty strings since there are no more lines in the file. One way to get back to the top of the file is to `close()` it and then `open()` it again.

f. **Looping through the input file object.** Create a new module to test this one. Again you need to first open() the file, and after you read from it, you should close() the file.

```
for <loop variable> in inputfile:
    print(<loop variable>)
```

will actually iterate through the *lines* of the file. So the above code would print each line one at a time, including the newline character at the end of each line.

(To understand better how this one works, note that it is equivalent in functionality to the following…)

```
for <loop variable> in inputfile.readlines():
    print(<loop variable>)
```

☺Get check 6☺

- Now let's try creating (writing) a new file:

  ```
  <output file variable name> = open(<file name>,"w")
  ```

  E.g., you might call your output file variable "outputfile" and wish to write to a file called lab2output.txt:

  ```
  outputfile = open("lab2output.txt","w")
  ```

- If the output file does not exist, it will be created with the file name specified. If it exists, then it will be overwritten with the new data.

- Now call the `write()` function as follows:

  ```
  outputfile.write(<string>)
  ```

  As a test, try writing your name (or any other phrase) for the string parameter.

- Call the `close()` command so that the file gets written/created. Then open it with Word or Notepad to check it out!

- Edit your code so that it writes a multi-line string to the file, using the character \n whenever you want to create a new line.

☺Get check 7☺

If you have more time: BONUSES

1. For part f of **check 6**, can you figure out how to get rid of the blank lines in between each line of the output?
2. In class we wrote a program that automatically generated a person's "Conn user ID" after they entered their first and last names. Modify the program so that it

will take as input a single full name from the user, eg. with a first name, several middle names, and a last name (for example: Mary Jane Doe Johnson). Modify the program so that it will generate a user name regardless of how many names they enter. The format of the user ID should be: first initial of each name, until the last name, from which at most the first 6 characters will be taken (eg. mjdjohnso for the example above). Hint: You might need to use the split() function to create a list of the name parts, then the len() function can be used not only to return the length of a string, but also the length of a list! Use a loop.

☺Get bonus checks (one part at a time in any order)☺

REMEMBER TO COMPRESS YOUR lab2_xxx folder and submit to Moodle.