22C:16 Homework 6

Due via ICON on Wednesday, March 21st, 4:59 pm

What to submit: Your submission for this homework will consist of three files: (i) `readWriteNumbers.py` for Problem 3, (ii) `samePowers.py` for Problem 4, and (v) `homework6.pdf` for the rest of the problems. Each of these Python files should start also with your name, section number and student ID appearing at the top of the file as Python comments. Also make sure that your Python code is appropriately commented. The pdf file should also start with your name, section numbers, and student ID appearing at the beginning of the file. You will get no credit for this homework if your files are named differently, have a different format, or if your files are missing your information.

1. Here is a simple problem that illustrates one way to read from a file in Python. To solve this problem, perform the following tasks:

   (a) Using a text editor such as `Notepad` (in Windows) or `TextEdit` (on the Mac OS) or `vi` (on unix), and create a file called `letter.txt`. It does not matter what `letter.txt` contains; just make sure that it has at least 2-3 lines and each line has at least 2-3 words.

   (b) Copy and save the Python program below in a file called `fileRead.py`.

```
f = open("letter.txt", "r")

for line in f:
    print line.split()[0]

f.close()
```

   (c) Run `fileRead.py`.

   Based on the output you see relative to what you typed into `letter.txt`, explain what the program does in 1-2 sentences.

   Answer: The program reads from the file called `letter.txt` one line at a time. It finds the string consisting of the leading non-whitespace characters in each line and prints this string, line by line.

2. This problem will illustrate one simple way of having a Python program produce output to a file. To solve the problem, perform the following tasks:

   (a) Copy and execute the following program:

```
fout = open("output.dat", "w")
fout.write("This is a test\n")
fout.write("And here is another line\n")
fout.close()
```

   (b) After you execute the program, you should see a file called `output.dat` in the same directory as the program. Open this file using a text editor.

(c) Now modify the program by deleting the `\n` at the end of each of the strings in Lines 2 and 3 of the program. Execute the program again and examine the output file produced.

Based on what you see, explain in 1-2 sentences what the `\n` at the end of each of the strings does.

Answer: The "`\n`" that appears at the end of each string in the write statements places an end-of-line character (or a return character) at the end of each string forcing the next string to appear in the next line. Without these "`\n`" characters, both strings that are written out will appear on the same line.

3. Write a program that reads from a file called "numbers.txt" and produces output to a file called "positiveNumbers.txt". Use the examples from the previous two problems to write code for reading from and writing to a file. The file "numbers.txt" contains one or more lines and in each line there are one or more numbers. Some of these numbers may be integers and the rest floating point numbers. Also, some of these numbers may be positive, some may be 0, and the rest negative. Your program should read "numbers.txt", pick out all positive numbers from this file and write these out to the file "positiveNumbers.txt". In the output file "positiveNumbers.txt" one number should appear in each line. Furthermore, the numbers in the output file should appear in the same order as they do in the input file.

   For example, suppose that "numbers.txt" is as follows:

```
3.7     -22 3500
   38   -11.993 2200  -1
3400 3400 -3400
-22
12 11 10 9.0
```

   Then after the program is executed, "positiveNumbers.txt" will be:

```
3.7
3500
38
2200
3400
3400
12
11
10
9.0
```

   Save this program in a file called `readWriteNumbers.py`.

4. Define a function called `samePowers` that takes a positive integer parameter n and returns a list of length n such that the element in position i is i^i. For example, if n were 4, the returned list would be `[1, 1, 4, 27]`.

   This is quite easy to do using a for-loop or a while-loop. For this problem, you are required to do it without using loops, instead using the `map` function. Save this function in a file called `samePowers.py`.

5. Consider the following program.

```
def strange(x, y):
    if x > y:
        return x - y
    else:
        return y - x

print reduce(strange, range(10))
```

(a) What output do you get when you run this program (you *are* allowed to just run it and find out)?

(b) Carefully explain why the program produces the output it does.

One way of thinking about how the `reduce` function "reduces" the list using the function `strange` is that it repeatedly send in as arguments to `strange` the most recent answer and the next element of the list. This gives the next answer. This evaluation is shown below and it gives the answer 5.

```
0 [1, 2, 3, 4, 5, 6, 7, 8, 9]
1 [2, 3, 4, 5, 6, 7, 8, 9]
1 [3, 5, 6, 7, 8, 9]
2 [5, 6, 7, 8, 9]
3 [6, 7, 8, 9]
3 [7, 8, 9]
4 [8, 9]
4 [9]
5
```
