COMP 361/651 Data Analytics Techniques in Python Assignment 6

7 possible points

Write a Python program that reads a file each line of which contains three values separated by any amount (but not none) of whitespace. There may also be any amount of whitespace (including, in this case, none) before the first value and after the last. The first value on a line should be in a floating-point format (explained in detail below), the second value should be a signed or unsigned decimal integer (see below), and the third value should be a string of three letters, upper or lower case, from the set {'x', 'y', 'z'}, repetitions allowed. You write a regular expression intended to match an entire line (so it begins with '^' and ends with '$'). Keep running sums of the floating-point numbers represented by the first values and of the integers represented by the second values in a line. Maintain a dictionary with the count of the number of 'x's, 'y's, and 'z's, upper or lower case, that have been encountered so far. If one or more of the values in a line are improperly formatted, output that line preceded by the string "No match with "; in this case, the entire line is ignored (making no contribution to the accumulated values).

The format for the integer value is simple: either (1) an optional sign followed by a string of digits beginning with a digit other than '0' or (2) the string consisting only of '0'.

For the format for floating-point value, there is an optional initial sign. Then, before the decimal point, there is either the character '0' or a string of digits beginning with a digit other than '0'. The remainder is a decimal point followed by one or more digits. The following are examples of valid floating-point representations.

```
12.34
-3.5
0.0
```

The following are examples of representations that fail to meet this format.

`12.` (There is no digit after the decimal point.)

`062.42` (The initial '0' is invalid.)

The following is the content of file **`data.txt`**, which you can download from the assignment site.

```
12.34  -79   xyz
-3.5 23    Yyz
32.2   2   zYx
4.23   +45  Zxz
62.456     -4   yxx
0.0    0   Zzz
12.0    -125    xYZ
062.42   12987  yZx
3.45  12.5      zzz
-45.2     06    xXx
43.6 72    yz
-2.63   -10    Xwz
21.4    2      xxyz
```

The last six lines are invalid. The first contains the invalid floating-point form starting with '0' just mentioned. The next two lines contain invalid integer representations (as the second values): `12.5` (not an integer) and `06` (invalid initial '0'). The third values in the last three lines are invalid: **`yz`** (the string must have three letters), **`Xwz`** ('w' not allowed), **`xxyz`** (the string must have three letters). The following is the output of the program using data.txt as input.

```
No match with 062.42   12987  yZx
No match with 3.45  12.5   zzz
No match with -45.2  06    xXx
No match with 43.6   72    yz
No match with -2.63   -10    Xwz
No match with 21.4    2      xxyz
Float sum: 119.726, Integer sum: -138
6 xs, 6 ys, 9 zs
```

To extract the values from the lines, in your regular expression, you will have to do grouping to capture sub-expressions (surrounding sub-expressions to be remembered with parentheses). When you need parentheses for grouping but not capturing, use non-capturing groups, which, recall, are of the form **`(?:...)`**. To update the dictionary of letters after each line, suppose that **`letts`** is the three-letter string matched by the third group in the regular expression and suppose that **`let_count`** is the dictionary with the count of the three letters. Then, for each value of **`i`** in [0, 1, 2], increment **`let_count[letts[i]]`** by one.

Use the verbose compilation flag, **`re.X`**. Break your regular expression into intuitive "units." Have one unit per line, and, with a comment, thoroughly explain what pattern is specified by that unit. (The general view is that your comments should not explain Python programming: assume that someone reading your code knows Python at least as well as you. Regular expressions, however, are different. We generally get a picture of the pattern and then work it out in detail as (part of) a regular expression. A well-commented regular expression can be read in much less time than one without comments.)