CGS600A: Computational Tools in Cognitive Science

Assignment #4: Python-3

Max marks:110
Due on/before:7 Sep.18, 23.59 27-Aug-2018

All programs are to be written in Python. Use functions to organize your programs logically.

1. A palindromic integer is a positive integer that is equal forwards and backwards. For example, 9, 33, 121, 11111, 209902 etc.

   You are give a positive integer n and a size m and you have to find the number of positive multiples of n whose length is m that are palindromic and the actual integers that are palindromic. For example, suppose n=12 and m=3 then there are 6 palindromic integers which are: 252, 444, 636, 696, 828, 888.

   Your program should read a sequence of n, m pairs written in a file - one pair per line and write an output file where for each pair it prints three lines: first the pair n, m second the number of palindromic integers and third the actual palindromic integers. So, if the input file has:

   ```
   12 3
   10 2
   ```

   The output should be:

   ```
   12 3
   6
   252, 444, 636, 696, 828, 888
   10 2
   0
      #blank line since there are no such palindromic integers
   ```

   [20]

2. You are given a file with the names of people one to a line. The full name may have words separated by one or more spaces. For example, 'Abhay Karandikar'. We want to classify the names into 3 categories 'Top heavy', 'Bottom heavy' and 'Balanced'.

   We check for heaviness as follows:

   1. Count the number of characters in the name that are in the first half of the alphabet (that is 'a' to 'm'), call this X. Similarly, count the number of characters in the second half of the alphabet (that is 'n' to 'z'), call this Y.
   2. if $X - Y > 1$ then 'Top heavy' else if $Y - X > 1$ then 'Bottom heavy' else if $|X - Y| < 2$ then 'Balanced'.

For a given input file with names write out an output file that contains in one line each name followed by its category and the values of X and Y.

Example:

```
Input file:

Abhay Karandikar
Aditi Jain
Aastha Sharma
Oviya Mohan
Axar Patel
Neetu Kapoor
Piyush Kurur

Corresponding output file:

Abhay Karandikar Top heavy 11 4
Aditi Jain  Top heavy 7 2
Aashtha Sharma Top heavy 9 4
Oviya Mohan Balanced 5 5
Axar Patel Balanced 5 4
Neetu Kapoor Bottom heavy 4 7
Piyush Kurur Bottom heavy 3 8
```

[25]

3. Use the *re* module in Python which implements a regular expression library for pattern matching to do the following:

   You have a file containing lines of text where the text can contain intervals of time in the format `<date_from><ws>to<ws><date_to` where `<ws>` is white space and `date_from` and `date_to` have the format `dd-mm-yy` if the date is in the twentieth century and `mm-dd-yyyy` if it is in the twenty first century. You have to create a file that contains the same text as in the input file except now all intervals are in the format `dd-mmm-yyyy<sp>to<sp>dd-mm-yyyy` where `<sp>` stands for space and `mmm` are the first 3 alphabetic characters of the month with first letter capitalized. So, 04 will be replaced by `Apr` and time intervals are not split over lines. They are always made part of the first line if it is split over two or more lines.

[30]

4. Consider finding the value of the following expression where each x_i, y_i is a positive integer and n can be very large:

$$P = \frac{x_1}{y_1} \times \frac{x_2}{y_2} \times \ldots \times \frac{x_n}{y_n}$$

   Assume our computer can only store integers between 1 and $2^{31} - 1$ (that is 32-bit integers with 1-bit for sign - however we are only interested in positive integers). Also, it is guaranteed that the numerator

and denominator of P in lowest form will be no larger than $2^{31} - 1$. Trying to calculate P by simply multiplying the x_is in the numerator and y_is in the denominator can easily lead to numbers much larger than $2^{31} - 1$. For example,

$$\frac{2}{5} \times \frac{2}{5} \ldots 50 \text{ times} \times \frac{2}{3} \times \frac{5}{2} \times \frac{5}{2} \ldots 51 \text{ times}$$

The value of the expression is just $\frac{5}{3}$ but straight forward calculation will lead to an overflow.

You have to read the sequence of fractions for each expression from an input file where each numerator-denominator is separated by ',' and each fraction is separated by white space. Each expression is separated by an empty line. You have to write the final output value for each expression in an output file - one value per line. For example, Input file:

```
1,2 3,4 9,6
13,8
4,5 8,3
2,13

36,7 8,5
7,9 5,4 2,24
```

Output file:

```
3,10
2,3
```

Note that n could be large, example 1000 so your program should handle cases where partial products can be larger than the maximum sized integer. [35]