Lab #9

Physics 91SI Spring 2013

Objective: This lab is designed to give you practice writing simple regular expressions and using them in Python scripts.

As usual, log on to `corn` and clone over the starter repository:

```
hg clone /afs/ir.stanford.edu/class/physics91si/src/lab9 lab9
```

Remember to `hg commit` often to save your changes, and submit your code at the end of the lab.

Part 1: Kodos Regex Exercises

Open the kodos program linked to from the starter repository (use `./kodos &`). This program provides a convenient graphical interface for writing and testing Perl/Python-style regexes. Start by entering some text in the middle window "Search String" tab. You can then write a regex in the top window, and the matches will appear in the "match" or "match all" tabs at the bottom.

To warm up, write regexes to match the following patterns:

- An email address, i.e. someone@somewhere.com
- A URL (http://www.website.com/page or similar)
- A name listed as lastname, firstname with the first letter of each capitalized
- A phone number, in the format of your choice
- A data in MM/DD/YY format (be careful about what values are permitted!)

You don't have to do all of these, but do enough so that you feel comfortable with regex syntax. For each example above, write down its corresponding regex in the file `part1.txt`, with each regex preceded by a 1-line description of what it does - you'll submit this file with the rest of the lab.

Part 2: Working with Groups

The files `people.txt` and `people-long.txt` contain lists of names, email addresses, and phone numbers in a consistent format; open one in a text editor to see what you're dealing with. Your task is to write a regular expression that allows you to selectively extract fields of interest while ignoring any extraneous information (like white spaces, formatting, etc.).

To do this, you'll want to create groups inside your regex using the `(pattern)` notation. Alternatively, you can name your groups with the syntax `(?P<name>pattern)`, which will assign each matched pattern a name. You can then extract the groups as a Python dictionary using `m.groupdict()`.

Use Kodos to prototype your regex, using a few sample lines from people.txt to test. Then, modify the starter code in `contacts.py` to extract the full name and email address of each person and print them in the following format:

```
John Doe: username@domain.com
```

Once you have this working, augment your code so that it prints them in alphabetical order (by first name). *Hint: if you have a list of tuples, the built in `sorted()` and `list.sort()` functions will sort by the first element.*

Submit your code if you've gotten this far - the last part is optional, and requires part of Lab #4 to work.

Part 3: Parsing a Dictionary

Do you remember writing `language.py` in lab 4 and how annoying it was to check if a word was valid? Your task now is to rewrite load_model() using regular expressions to find valid words. You'll need the same code to add them to the dictionary, but you can use `re.findall()` or `re.finditer()` to more easily scan the file and remove punctuation.

To grab your copy of language.py and bring it over to lab 9, you'll need to first submit lab 4 (don't worry if you haven't finished it). Then go back to the lab 9 directory and run `python init.py` to automatically copy over language.py from your submission and add it to the repo. (init.py is a Python script that runs some shell commands - take a look at it if you're interested)

Once you've rewritten load_model(), go ahead and add some more regex goodness to your program. Some ideas:

- Search the dictionary for hyphenated words
- Build a dictionary of hamlet.txt and compile a list of the characters (they're in all caps)
- Compile a dictionary of only those words found at the beginning of a sentence, or preceding a semicolon, or other contextual filter

Also, if you're feeling gutsy, there's a way to make `load_model()` even more elegant. The

Python standard library provides a `Counter` class designed for counting occurrences - exactly what you're doing with the language model. Look up the documentation, and access the class by adding `from collections import Counter` to the top of your .py file. Note that it takes a list - exactly what you get from `re.findall()`!