Ling 5801 – Introduction to Computational Linguistics (S20, Reese)
Assignment 1: Information extraction using regular expressions
Due Monday, February 17, 2020.

General instructions : Assignments require you to use specific names for the files that you submit. Included with assignments are "stub" files containing instructions and hints, which you can download for Canvas and edit. **Do not change the names of these files!**

Your task is to write a Python regular expression that searches documents for **date expressions**.

This might seem like a straightforward task, but there are many different conventions for formatting dates (e.g., "month-day-year", "day-month-year" and "year-month-day") and for each of these conventions there are number of additional variables to consider (what is the separator? is the month given numerically? is the month name abbreviated? etc.). We have deliberately obfuscated the documents that you will search so that there is no one consistent format for date expressions. This is not an unrealistic situation. For example, if we were processing a large collection of documents, it would be very likely that different documents adopted different conventions for dates. Therefore your regular expression needs to match as many of these patterns as possible without mistakenly matching non-date expressions.

The file `dev_gs.txt`, the "gold standard", contains the full list of date expressions from the development text `dev.txt` that your regular expression should match. **Inspect this gold standard file to get a sense of which patterns your regular expression should be sensitive to.**

Once again, your task is to write a regular expression. More detailed instructions and background follow.

Instructions:

1. Download the files `DateFinder.py`, `dev_gs.txt` and `dev.txt` from Canvas and save them in the same folder. The file `dev.txt` is the development text; this is the document that your regular expression will search for date expressions. `dev_gs.txt` is a list of date expressions in the development text that your regular expression should match. You will use this list to evaluate the output of your regular expression.

2. `DateFinder.py` contains a string named `pattern`. This is where you will define your regular expression. **This is the only part of `DateFinder.py` that you need to (and should) edit!**

   We will compile `pattern` using the `re.VERBOSE` flag so that your regular expression can span multiple lines and include comments. This is useful when defining complex patterns. For example, you could have each line of your regular expression represent a different way to format a date and combine each pattern using disjunction.

   However, how your regular expression is formatted is ultimately up to you. This way of compiling a regular expression ignores whitespace. Therefore if you want to match a space you will need to escape it by putting a backslash in front of it. Alternatively, use the `\s` shorthand which matches *any* whitespace character.

3. If you use parentheses in your regular expression, which is likely given the complexity of the pattern, you need to use the non-capturing flavor of parentheses – `(?: ...)` – in order to ensure that your output is correctly formatted.

4. The Python code in `DateFinder.py` gives your pattern and the contents of `dev.txt` to the `re.findall()` method, which returns a list of the strings in the development text that your regular expression matches. The resulting list is then evaluated against the gold standard list of dates using the precision, recall and f-score metrics discussed in class.[1]

   Keep in mind that **recall** rewards correct guesses, **precision** penalizes incorrect guesses, and **f-score** combines precision and recall. As an illustration, `DateFinder.py` includes an initial definition for `pattern` that simply looks for three groups of one or more numeric characters with any single character occuring between them.

   ```
   pattern = r"""
       \b\d+.\d+.\d+\b  # a simple pattern
       """
   ```

   The scores for this pattern are as follows.

   ```
   Precision: 0.450000          Recall: 0.290323          F1: 0.352941
   ```

   It only mathced 29% of the dates in the development text and 45% of the strings it returned were actual date expressions.

5. **Your grade is your F1 score!** However, we will not evaluate your regular expression against the development text, but rather against an unseen test text. The test text will contain exactly the same date *formats* as found in the development text, so if your regular expression is general enough, your score on the test text should be similar to your score on the development text.

6. When you are satisfied with the performance of your regular expression submit `DateFinder.py` via Canvas. Submissions are due by 1pm on Monday, February 17.

Start working on the assignment right away and if you have questions, please ask them sooner rather than later. Good luck!

[1]Note that if your pattern does not match any strings, the evaluation code will throw an error due to a division by zero. This means that your pattern isn't matching any strings and the list `dates` is empty.