Quiz #2: EBNF and Regular Expressions ICS-33 Spring 2014

Name ____________________

Lab # (1-4) ____________________

Recall the Academic Integrity statement that you signed. Write all answers clearly on these pages, ensuring your final answers are easily recognizable. The number of points for each problem is clearly marked, for a total of 25 points. I will post my solutions on the web on Wednesday, off the **Solutions** link, after class.

Download the `q2helper` project folder (available for Friday on the weekly schedule) in which to write/test/debug your code for problem 4. Submit your completed `q2solution.py` module online and write your answers on this quiz (hand in only one two-sided sheet: not multiple pages attached/stapled). Both are due at the start of class on Wednesday.

1. (6 pts) In English, we often write one name by itself (`Bob`), two names with just an `and` between them (`Bob and Carol`), and more than two names with commas between them, with the last name also preceded by an `and` (`Bob, Carol, Ted and Alice`). Notice there is no `,` before the `and`.

Write an EBNF description named ***name-sequence*** that specifies only legal (according to the previous description) sequences of names (one, two, three, or more names). Note writing `Bob, Bob and Bob` is allowed. Your solution should comprise a single big EBNF rule. Try to write the simplest, most elegant solution possible. Use the ***name*** rule below in your ***name-sequence*** rule, along with `,` and `and`. Your rule must specify legal combinations, of one, two, three, four, … any number of names by the pattern above and not specify illegal combination: e.g., `Bob, Carol` (two names requires `and`), `Fred and Barney and Wilma` (only one `and` allowed), etc.

name ⇐ `Bob | Carol | Ted | Alice | Fred | Wilma | Barney | Betty`

name-sequence ⇐ ***name*** `[{,`***name***`} and` ***name***`]`

2. (4 pts) Translate each of the following patterns (EBNF to Regular Expression or Regular Expression to EBNF) . Try to write the simplest equivalent translation.

EBNF From	Regular Expression Form
(a) `[+|-]`	`[+-]?`
(b) `a{a}`	`a+`
(c) `a[{xa}ya]`	`a((xa)*ya)?`
(d) `e{e}{bb|p[p][p][p]d}c`	`e+(bb|p{1,4}d)*c`

3. (6 pts) Write a regular expression pattern that matches dates written in the format **month/day/year**. Here **month** can be any one- or two-digit number 1-12 (with no leading 0 allowed), **day** can be any one- or two-digit number number 1-31(a leading 0 is allowed: **05** is OK), **year** that is any two- or four-digit number (leading 0s are allowed; a two-digit date is in the 21st century: **00** means **2000**).

The `q2helper` project folder contains a `bm.txt` file (examine it) to use for batch-matching your pattern, via the `bm` option in the `retester.py` script. These are rigorous but not exhaustive tests. Here are a few selected examples from this file. Hint: my solution pattern was 55 characters, ending in a `$`; it generates exactly 3 matching groups for the month, day and year: I used `?:` in once where I needed parentheses for grouping but not for a matching group.

Match: `1/1/2000`, `1/1/00`, `1/01/00`, `1/31/00`, `2/31/2000`, `1/1/0000`

Not Match: `05/1/00`, `13/1/00`, `2/005/00`, `1/32/00`, `1/1/200`, `1/1/20000, 5/0/2000`

Write your answer here, and submit it on Checkmate (so we can cute/paste and test it in `retester.py`).

```
([1-9]|1[0-2])/(0?[1-9]|[1-2]\d|3[01])/((?:\d\d)?\d\d)$
```

4. (9 pts) Define a function named `tomorrow` that takes one string as an argument, representing a date (use your regular expression from problem #3); it returns a string representing one date later. This function **must use a regular expression pattern** to extract (using **groups**) the **month**, the **day**, and the **year** of the date. Calling `tomorrow('9/4/1988')` returns the string `'9/5/1988'`; `tomorrow('9/30/1988')` returns the string `'10/1/1988'`; `tomorrow('12/31/1988')` returns the string `'1/1/1989'`; etc. Note that the result returned should have **no leading 0s** for the **day** and always have a **four-digit year**. If the pattern does not match or the number of days in the month is not legal (e.g., 31 days in February; this constraint is not handled by the regular expression) raise an `AssertionError` exception (using Python's `assert` statement) with appropriate information. Handle leap years correctly for February (using the `day_dict` and `is_leap_year` functions written in the `q2solution.py` module).

Hint: use `re.match` and use **groups** to extract the **month**, **day**, and **year** and then increment the **day**, resetting the **day**/incrementing the **month** if necessary, and resetting the **month**/incrementing the **year** if necessary. My function was 12 lines of code.

The `q2helper` project folder contains a `bsc.txt` file (examine it) to use for batch-self-checking your function, via the `driver.py` script. These are rigorous but not exhaustive tests. If you did not correctly solve for the pattern in part 2, use `(\d+)/(\d+)/(\d+)$` here, which will allow you to extract **month**, **day**, and **year** groups (even if the date is malformed).

Do not write your answer here, but include this code in the `q2solution.py` module submitted on Checkmate.