When working on this quiz, recall the rules stated on the Academic Integrity statement that you signed. You can download the **q2helper** project folder (available for Friday, on the **Weekly Schedule** link) in which to write your Regular Expressions and write/test/debug your code. Submit your completed files for **repattern1a.txt**, **repattern1b.txt**, **repattern2a.txt**, and your **q2solution.py** module online by Thursday, 11:30pm. I will post my solutions to EEE reachable via the **Solutions** link on Friday morning.

For parts 1a, 1b, and 2a, use a text editor (you can use Eclipse's) to write and submit a one line file. The file should start with the **^** character and end (on the same line) with the **$** character. The contents of that one line should be exactly what you typed-in/tested in the online Regular Expression checker.

The **q2helper** project folder contains a **bm1a.txt**, **bm1b.txt**, and **bm2a.txt** files (examine them) to use for batch-matching your pattern, via the **bm** option in the **retester.py** script. These patterns are also tested automatically in the **q2helper**'s **bsc.txt** file (using similar examples to those in the **bm.txt** file).

1a. (2 pts) Write a **regular expression** pattern that matches times on a 12-hour clock written in the format **hour:minute:second**. Here **hour** can be any one- or two-digit number in the range 1-12 (with no leading 0 allowed); **minute** is optional: if present it can be any two-digit number in the range 00-59; **second** is optional: if present, it can be any two-digit number in the range 00-59; at the end is a mandatory **am**/**pm** indicator. Here are a few legal/illegal examples.

Legal: Should Match : `6pm`, `6:23pm`, `6:23:15am`, `12am`, `11:03am`, `8:40:04pm`

Illegal: Should Not Match: `6`, `06pm`, `14pm`, `6::pm`, `6:60pm`, `6:111pm`, `6:4pm`, `6:04:7pm`, `6:23:15:23am`

1b. (3 pts) Write a **regular expression** pattern that matches integers according to the following rules: `0` is an integer. Any integer except `0`can start with a plus or minus sign. The first digit of any integer, other than `0`, must be non-`0`. An integer can have any number of digits in it, but if it has four or more digits, commas must occur in the correct positions. Here are a few legal/illegal examples.

Legal: Should Match : `0`, `+1`, `-10`, `+102`, `1,024`, `-10,427,485`

Illegal: Should Not Match: `+0`, `-0`, `05`, `1,0`, `1,02`, `1,,024`, `1024`, `1024,`

2a. (5 pts) Write a **regular expression** pattern that matches dates according to the following rules: date are written in three parts, in the order **month**, **day**, and optional **year** where the parts are separated by the **/** character. The **month** is a one or two digit number in the range `1-12` (with no leading `0` allowed). The **day** is a one or two digit number in the range `1-31` (with a leading `0` allowed); it can also be a single space followed by a one digit number. The **year**, if included, is a two digit number (each digit can be **0**: **00** means **2000**). Here are a few legal/illegal examples.

Legal: Should Match : `2/10`, `2/10/06`, `12/31/15`, `12/ 3`, `12/03`, `2/31`, `9/ 4/13`

Illegal: Should Not Match: `02/10`, `13/10`, `21/13`, `12/ 13`, `5`, `5/`, `5//`, `5/12/2016`

2b. (10 pts) Define a function named **future_date** that first takes a string as an argument (representing a possible date: see 2a for the specification of legal dates); its second argument is the number of days to "advance" the date into the future (`5` means 5 days into the future), and its third argument is the current year (e.g., `2016`). This function returns a string representing the future date. For example, calling the function `future_date('3/27/16',10,2016)` returns the date 10 days in the future; it returns the value `'4/6/16'`; calling `future_date('3/27',10,2016)` returns `'4/6'`: if no year is specified, use the 3rd argument as the year, but don't include the year in the returned value (even if it changes). The **year**, if present, must always appear in the returned result as two digits : years `2000` or `2005`, would appear as `'.../00'` and `.../05'`.

Raise an **`AssertionError`** exception (using Python's **`assert`** statement) if the first argument is an illegal date: by the specification in 2a, or if the actual day specified is not legal in the month specified (e.g., **2/30** in any year). Also raise this exception if the second argument is negative (0 means no advancement).

Hint: Call **`re.match`** on the **regular expression** pattern from 2a and the first argument. After determining the integers representing the **month**, **day,** and **year** (using groups, conversion functions, and logic – and possibly the third argument) repeatedly "advance" to the next date the number of times specified by the second argument. Advance by one **day**, going to the next **month** if advancing beyond the end of the month, and going to the next **year** if advancing beyond the end of the year. Then take the results and compute the correct string to return. Note that **`x%100`** returns the last two digits in **`x`** (do you know why?). Note that the value returned when an option (using **`?`**) matches no characters is **`None`** My function body was 23 lines long.

3. (5 pts) Write a function named **`compare files`**, that takes two open files as arguments. It returns a list of 3-**tuples**: it reads each file, comparing 1st lines, 2nd lines, etc. Each 3-**tuple**, in the list it returns, contains the line number for lines that are different, followed by the line from the first file and line from the second file (with the newline stripped from the ends of the file lines). The list stops after comparing the last line of the shorter file. For example if the files **`cf1a.txt`** and **`cf1b.txt`** store the information shown below, then calling the function **`compare files( open('cf1a.txt'), open('cf1b.txt') )`** returns **`[(2, 'b', 'x'), (4, 'd', '?'), (5, 'e', '?')]`**.

Hint: I wrote my function as a single list comprehension, with one loop, using parallel assignment and various useful Python built-in functions.

cf1a.txt

```
a
b
c
d
e
```

cf1b.txt

```
a
x
c
?
?
y
z
```