CIS 480 - Python - Fall 2005
WEEK 9 LAB EXERCISE and Homework #7

week 9 lab exercise due: Thursday, October 20th, END of lab
HW #7 due: Thursday, October 27th, 12:00 noon

Until I develop more formal opening comment block standards (which I STILL plan to do), begin **each python module** that you write with at least the following opening comments:

* a comment containing the name of the module (the **file name** of the module, please --- **hw07.py**, for example)
* a comment containing your name, and
* a comment containing the date that your module was last modified

WEEK 7 LAB EXERCISE

To be done in-lab, "lightening" give-an-RE-string questions as we go through a few more re module tidbits.

HOMEWORK #7

1. First, create a file **hw07_patterns.txt**. Within it, write a Python RE string that meets the criteria given for each part below. (Note that there is nothing keeping you from trying out your patterns to test them, either using **re_play.py** from lecture or some other testing code that you design...)

Note: I should be able to plop your answers, **as you give them**, into a call to **re.compile** and have them match as specified (write them as an appropriate kind of Python string literal...!)

Precede each pattern with the letter of the part it is an answer to...!

(a) Write a pattern that will match any string which includes amongst its characters 3 or more consecutive digits.

(b) Write a pattern that will match any string which includes amongst its characters 3 or more digits --- and those digits need NOT be consecutive.

(c) Write a pattern that will match any string that includes amongst its characters the substring 'Ni'.

(d) Write a pattern that will match any string that includes amongst its characters the 'word' 'Ni' --- that is, I do not want 'Nightingale' to match, nor 'niNinny'. (Yes, I want you to write a pattern making proper use of \b...)

(e) There is a music term, **fortissimo**. (double-f, or ff --- loud!) **fortississimo** is even louder (triple-f, or fff), and it can continue from there, to the ridiculous (fortississsisssisssissimo, fffffff ...)

Write a pattern that will match fortissimo, or fortississimo, or fortississsimo, and so on!

(f) Write a pattern that will match any word that begins with a vowel (defined to be a, e, i, o, u, or A, E, I, O, U). (The word should only contain letters, and begin with one of those particular vowel letters.)

(g) Write a pattern that will match any word that DOESN'T begin with a vowel (defined still to be a, e, i, o, u, or A, E, I, O, U). (The word should only contain letters, and begin with a NON-vowel letter.)

(h) Write a pattern that will match any string that includes amongst its characters the character **<** followed by any number of characters followed by the character **>** (that is, the < and > can have anything in-between, and maybe nothing in-between). (That is, your pattern should match any string that includes at least one pair of angle brackets --- at least one < followed by at least one >).

(i) Write a pattern that will match any string that includes amongst its characters the character **<** and the character **>**, in ANY order. (it can have more than one < and more than one >, but it has at least one, and they should be able to appear in ANY order.) (Note that this pattern should NOT be the same as (h)'s pattern!)

(j) Write a pattern that will match any string containing a backslash character amongst its characters.

Now, for problems #2 - #5, create a Python module **hw07.py**. Within it, include the following:

2. Write a predicate **has_3letter_word** that expects a string as an argument. It returns bool value **True** if the string contains a `\b`-delimited substring containing exactly 3 letters (a-z, A-Z), and bool value **False** otherwise. (By `\b`-delimited, I mean that I want you to use `\b` in your pattern so that you don't match, for example, 4-letter words or longer.)

(That is, a word with 4 or more letters should **not** satisfy this; you need to look for **word boundaries**, too. We'd like for a string of 3 letters to satisfy this, though.)

In a separate module **hw07_test.py**:

* import the **hw07.py** module,
* write a **print** statement that says, **testing has_3letter_word**.
* write a **print** statement that says, **"True == Passed, False == Failed"**
* and then write print statements comparing calls to **has_3letter_word** to its expected value.

For example,

```
print "Testing has_3letter_word:"
print "   (True == Passed, False == Failed)"
print "-----------------------------------"
print hw07.has_3letter_word("") == False
print hw07.has_3letter_word("dog") == True
print hw07.has_3letter_word("123") == False
print hw07.has_3letter_word("a good day to ride") == True
print hw07.has_3letter_word("good riddance") == False
```

...for testing calls involving at least the above tests and any additional tests you'd like to include.

3. Now, consider a **telephone number**. Let's define a telephone number as containing **7** digits, possibly with a space or a dash between the first three digits and the last four digits.

Write a function **grab_first_phone**. It takes a string as its argument, and returns the first phone number it finds (according to the above definition of a phone number). It returns that phone number in the form of a string, or it returns the NoneType value None if there is no phone number in the string. We'd like it to be somewhat discriminating in that a sequence of more than 7 digits should not match this --- again, we'd like the phone number to be a `\b`-delimited substring of the string.

Test it in **hw07_test.py** as you tested the function from problem #1; test it on at least:

* a string containing a proper phone number all by itself,
* a string containing a phone number with a dash within,
* a string containing a phone number with a blank within,
* a string containing a phone number with neither dash nor blank,
* a string containing a phone number and other stuff,
* a string containing no phone number,
* a string containing a sequence of 8 or more digits (that should not, then, be considered a phone number)

...and any additional tests you would like to include.

+10 BONUS: Have this look for an optional area code, too; this optional area code might or might not be surrounded by parentheses, and it might be separated from the main number by a blank, or it might be separated from the main number by a dash, or it may not be separated from the main number at all.

For full credit, **hw07_test.py** should also include at least the tests already mentioned above (except modify the last required test appropriately...!), PLUS:

* a string containing a phone number with an area code surrounded by parentheses,
* a string containing a phone number with an area code NOT surrounded by parentheses,
* a string containing a phone nunber with an area code separated from the main phone by a blank,
* a string containing a phone number with an area code separated from the main phone by a dash,
* a string containing a phone number with an area code NOT separated from the main phone,
* a string containing a sequence of 11 or more digits (that should not, then, be considered a phone number)

4. Note that **grab_first_phone** lets you know if there is any phone number (and if so, it returns the first one found). What if there is MORE than one phone number in the string?

Write **grab_all_phones** that returns a list of all of the phone numbers in its string argument... if there are none, it should return an empty list. (You can use your pattern from **grab_first_phone** --- so, if you did the bonus there, you can use that same area-code-including pattern here too, if you'd like.)

For **grab_all_phone**'s tests in hw07_test.py, include at least three test calls each containing a different number (each greater than 1) of phone numbers, and at least one test call containing no phone numbers.

5. Now, write a function **mask_phones** that takes a string assumed to contain a file name in the current directory as its argument, and then creates a file in the current directory whose name is the original name preceded by "**masked_**" that contains the original file's contents, EXCEPT every phone number within it has been replaced with **XXX-XXXX**.

For example, say that a file **tryme.txt** in the current directory contained the following:

```
I think I'd like to call 867-5309.
Or maybe 826-3381, or 825-7727. Wonder if the commas will freak it out?
Oh dear --- it might, at that! 123-4567 890-5544 123 4567
Is that enough?
```

After running `hw07.mask_phones('tryme.txt')`, there will now be a directory named **masked_tryme.txt** in the current directory, containing:

```
I think I'd like to call XXX-XXXX.
Or maybe XXX-XXXX, or XXX-XXXX. Wonder if the commas will freak it out?
Oh dear --- it might, at that! XXX-XXXX XXX-XXXX XXX-XXXX
Is that enough?
```

You get a choice: you can make use of **grab_all_phones**, and then it is your choice if you'd rather use just Python string class methods to proceed or use additional **re** stuff. OR, if you do not make use of **grab_all_phones**, then you must use **re** stuff in your solution.

This being file-input-output related, I'm not sure how to have you test it in **hw07_test.py**. I'll just have to test your function on a file of my own creation... 8-)

And, when you are satisfied with the above problems, you should create a submittable output file **hw07_test.out**:

```
python hw07_test.py > hw07_test.out
```

By the due date and time given at the beginning of this handout, use **~st10/480submit** to submit your final versions of **hw07_patterns.txt, hw07.py**, **hw07_test.py**, and **hw07_test.out** (And remember --- you can submit more than one version of these before the deadline, if inspiration strikes after a submission. As the syllabus notes, I'll simply grade the latest version that was submitted before the deadline.)