COMS 3101-3 Programming Languages – Python
Assignment 4 (25 points)
Due: 2PM, Oct 10th 2013

Each part of the assignment should be in placed under directory named 'part1/', '/part2', and '/part3'. Please submit a compressed file (zip or gz) including your source and README to Courseworks. The file should be named as [your_uni]_hw[X].zip where X is the number of homework.

Part1. (5 points) Using Python's regular expression (re) module, write functions that examine a *string* input and return Boolean value (True or False) satisfying conditions stated below. For submission, create a module 'part1' to have all your functions in it.

(a) Create a function `is_alpha(str0)`: which returns `True` if all characters are alphabetic and there is at least one character in `str0`, `False` otherwise. Thus, the function has to be equivalent to `str.isalpha()` method.

(b) Define function `is_alnum(str0):` which returns `True` if all characters are alphanumeric and there is at least one character in `str0`, `False` otherwise. Thus, the function has to be equivalent to `str.isalnum()` method.

(c) Define function `startswith(str0, prefix)` Return `True` if `str0` starts with the specified prefix, False otherwise Thus, the function has to be equivalent to `str0.endswith(str0)`method.

(e) Define function `is_in(str0, str1)`: Returns `True` if `str1` is a substring of `str0`, `False` otherwise. This function has to be equivalent to `'in'` operator. i.e, `is_in(str0, str1)` is functionally e str1 in str0.

Part2. (10 points) In this part, we will write a simple web crawler using regular expressions. We will use the urllib2 module to retrieve web documents.

(a) Write a program that accepts a start web address (*URL*) and a recursion depth *d* as command line arguments and prints list of URLs visited as follows.

Basic usage:

```
$ python crawler.py URL d
```

The program should read in the web document at the start address, extract all web links that are mentioned in the document using regular expressions, and then continue the process recursively for each link until d recursion levels have been reached. If the program is called with recursion depth 0, only the document at the start address should be retrieved.

The relevant HTML links are enclosed in a `<a href></a>` tag and it can be in either absolute or relative form.

Absolute link:

```
<a href="http://server/pathname">link description</a>
```

Relative link:

```
<a href=" pathname">link description</a> # relative to current location
<a href=" /pathname">link description</a> # relative to remote server
```

Note that HTML code is not case sensitive, both single and double quotes can be used and whitespaces (outside the actual address) are arbitrary. Your crawler should print out *the titles and addresses* of all pages it visits. It should print a warning message if a page cannot be opened. Make sure your crawler does not run into cycles (i.e. it should ignore pages it has already seen).

```
$ python crawler.py http://www.cs.columbia.edu/~jikk/index.html 2
Title: Kangkook home    URL: http://home.com/~jikk/index.html
Title: Publications     URL: http:// home.com /~jikk/pubs.html
...
```

Hint: Before implementing the crawler, write good regular expressions and test them on tricky example cases such as:

```
<A Href= 'HTTP://s1.s2.DOMAIN.EDU:80/˜usr/script.cgi?foo=bar&answer=4'>
```

(b) Extend your crawler script to build unit testing framework to validate the correctness functions that you wrote from part 1.For every title string the crawler gets from its visits, it should run test-cases to see whether the following invariants hold,

```
is_alpha(title_string)    <==> title_string.is_alpha()
is_alpnum(title_string)    <==> title_string.is_alnum()

#for all words in title_string
startswith(title_string, word) <==> title_string.startswith(word)

#for all words in title_string
is_in(title_string, word) <==> word in title_string
```

Part 3. (10 points) In this problem, we will use the 'pickle' module to save and load secret messages hidden in inconspicuous Python objects.

(a) Write a module `'gen_key'`, that contains a class `Key` with base class `dict`. This class represents a simple substitution cipher. When Key instances are initialized they should randomly map each of the 26 English letters [A-Z] and whitespace to a unique ciphertext number between 0 and 26. Write a main function that creates a Key instance and uses pickle (or cPickle) to save it to a file (specified on the command line). Note that all modules that will unpickle a Key will need to import the Key class from the `'gen_key'` module.[1]

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
0	24	18	23	2	15	14	10	13	1	11	17	4	21	22	20	6	3	19	12	8	16	9	25	7	5	26

(b) Write a module `'encode'`, that contains a function encode, which takes as parameters a plaintext message and a key instance and encodes the message in the following simple unary representation. The message is represented as a list and each element of the list represents a letter. Each letter is a list of objects of any type, such that the length of the list represents the ciphertext number for the letter (according to the key).

For instance, the message 'MARY' encoded with the key above would look like this:

```
[
  ['a', 'a', 'a', 'a'],                          # M : 4
  [],                                            # A : 0
  ['a', 'a', 'a'],                               # R : 3
  ['a', 'a', 'a', 'a', 'a', 'a', 'a'],           # Y : 7
]
```

Write a main function that reads in a plaintext message from the keyboard, uses encode to encode the message, and then uses pickle (or cPickle) to save the encrypted message into a file. Output file and Key should be passed as command line arguments.

(c) Write a module `'decode'`, that contains a function decode, which takes as parameters an encoded message object, of the type described above, and a Key object, and returns a plaintext string. Write a main function that *unplickes* a message object and a Key object from a file, uses decode to decode the message and print the plaintext. Input file and the pickle file containing the Key should be passed as command line arguments.

[1] Look at the `shuffle(sequence)` function in the `'random'` module. Use the function `zip(seq1, seq2)` to do this in two lines.