Homework 1: Spamlord [starter code]

Harvesting emails and phone numbers

This assignment is your chance to become a Dark Lord of spam email! Yes, you too can build regexes to spread evil throughout the galaxy.

More specifically, your goal in this assignment is to write regular expressions that extract phone numbers and regular expressions that extract email addresses. We recommend working on the phone number questions first as they are simpler and a good warm-up for the email regexes.

To start with a trivial example, given

```
manning@cs.stanford.edu
```

your job is to return

```
manning@cs.stanford.edu
```

But you also need to deal with more complex examples created by people trying to shield their addresses, such as the following types of examples that I'm sure you've all seen:

```
manning(at)cs.stanford.edu

manning at csli dot stanford dot edu
```

You should even handle examples that look like the following (as it appears on your screen; we've used metachars on this page to make it display properly):

```
<script type="text/javascript">obfuscate('stanford.edu','manning')</script>
```

For the three examples above you should return the corresponding email addresses:

```
manning@cs.stanford.edu

manning@csli.stanford.edu

manning@stanford.edu
```

as appropriate.

Similarly, for phone numbers, you need to handle examples like the following:

```
Phone:  (650) 723-0293

  Tel (+1): 650-723-0293

  <a href="contact.html">TEL</a> +1&thinsp;650&thinsp;723&thinsp;0293
```

all of which should return the following canonical form:

```
  650-723-0293
```

(you can assume all phone numbers we give you will be inside North America).

In order to make it easier for you to do this and other homeworks, we will be giving you some data to test your code on, what is technically called a *development set*. This is a document with some emails and phone numbers, together with the correct answers, so that you can make sure your code extracts all and only the right information. You don't have to worry about case sensitivity in your output emails, since they will be normalized to lower case before being compared with the answers.

You will be graded on how well your regular expressions find emails and phone numbers in a different *test set* that we have. Because you don't know exactly what trickery goes on in this test set, you should be creative in looking at the web and thinking of different types of ways of encoding emails and phone numbers, and not just rely on the methods we've listed here, or are listed in the homework.

You won't have to deal with: really difficult examples like images of any kind, or examples that require parsing names into first/last like:

```
  "first name"@cs.stanford.edu
```

or difficult examples like my friend Jim Martin, whose email was listed only as:

```
  To send me email, try the simplest address that makes sense.
```

Note: we recommend working on the phone number questions first as they are simpler and a good warm-up for the email regexes.

Where can I find the starter code?

We have provided starter code written in Python 3.5.2, that you can download here . You can transfer these files to rice by using the following command **on your local computer**, which will prompt you to sign in:

```
  $ scp -r <local path to starter code> <sunetid>@rice.stanford.edu:<path to
desired location on rice>
```

The directory structure looks like:

```
  spamlord/

    data/

      dev/ # The development set

        aiken

        ashishg

        ...

      devGold # What we think are the right answers

    python/

      __init__.py

      SpamLord.py # python starter code
```

By default, if you execute:

```
  $ cd python

  $ python3 SpamLord.py ../data/dev/ ../data/devGOLD
```

in the python directory, it will run your code on the files contained in data/dev/ and compare the results of a simple regular expression against the correct results. The results will look something like this:

```
  True Positives (4):

  set([('balaji', 'e', 'balaji@stanford.edu'),

       ('nass', 'e', 'nass@stanford.edu'),

       ('shoham', 'e', 'shoham@stanford.edu'),

       ('thm', 'e', 'pkrokel@stanford.edu')])

  False Positives (1):

  set([('psyoung', 'e', 'young@stanford.edu')])

  False Negatives (113):
```

```
set([('ashishg', 'e', 'ashishg@stanford.edu'),
     ('ashishg', 'e', 'rozm@stanford.edu'),
     ('ashishg', 'p', '650-723-1614'),
     ('ashishg', 'p', '650-723-4173'),
     ('ashishg', 'p', '650-814-1478'),
...
```

The true positive section displays e-mails and phone numbers which the starter code correctly matches, the false positive section displays e-mails which the starter code regular expressions match but which are not correct, and the false negative section displays e-mails and phone numbers which the starter code did not match, but which do exist in the html files. Your goal, then, is to reduce the number of false positives and negatives to zero.

Where should I write my Python code?

The function

```
def process_file(name, f):
```

has been flagged for you with the universal "TODO" marker. This function takes in a file object (or any iterable of strings) and returns a list of tuples representing e-mails or phone numbers found in that file. Specifically the tuple has the format:

```
(filename, type, value)
```

where type is either 'e', for e-mail, or 'p' for phone number, and value is just the actual e-mail or phone number.

What version of Python should I use?

In this class, we will be using Python 3.5. Please see this document (Links to an external site.) for tips on how to set up Python 3.5.

What format should the phone numbers and e-mails have?

The canonical forms we expect are:

```
user@example.com
```

```
650-555-1234
```

The case of the e-mails you find should not matter because the starter code and the grading code will lowercase your matched e-mails before comparing them against the gold set.

How will I be graded?

This assignment will be graded out of 24 points. The performance of your system on the dev set will count for 16, and the performance on the test set will count for 8. Here is the algorithm we used for each section where e is the total number of errors (false negatives and false positives) for that data set.

dev:

```
if e < 10 then score(e) = 16 - e

else if e >= 10 then score(e) = 6
```

test:

```
if e <= 10      then score(e) = 8

else if 10 < e < 20 then score(e) = 13 - .5 * e

if e >= 20      then score(e) = 3
```

Submitting Your Solution

Note: make sure your submission works on the cluster virtual environment as this is the environment our autograder will score you on.

Submit your assignment via **Gradescope** (www.gradescope.com). You can create a submission with the files below, and it will run the remaining grading with the starter code. We expect the following files in your final submission:

- SpamLord.py (do not alter the filename)

If your solution depends on other files, please put those files in a folder named "deps/" (this folder is contained in the python directory so it is on the same level as SpamLord.py) and upload a zip file (arbitrarily named) containing this folder and SpamLord.py to submit on Gradescope. Gradescope will then automatically unzip the folder so that your submission contains:

- deps/

- SpamLord.py

For now we are only showing the score on the dev set (16 possible points). The test set will be evaluated after the submission deadline (8 possible points).