# 11-4/611 Assignment 1: Warming up with Regex

Assigned: 29 August 2019
Due: 5 September 2019 11:00PM
unlimited submissions

This is an easy introductory assignment to gauge your Python programming abilities. It is designed to warm you up with some basic text processing in Python. This assignment is worth 100 points in total.

## 1 Files

Download the assignment package from Piazza. You should have the following files when you uncompress the folder.

```
working directory
├── nyt
│   ├── file0.txt
│   ├── file1.txt
│   └── ..
├── cleaner.py
├── names.txt
├── decode.py
├── encoded.txt
└── key.txt
```

## 2 Background

The year is 1984. Your name is Winston Smith. You live in Airstrip One and work for the Ministry of Truth. You have been assigned the important task of making sure that the media only reports stories that are true; that is, that agree with the Inner Party's official historical records.

# 3 Part 1: Name Censorship (70 points)

It has come to the Party's attention that one particularly suspect media outlet, the New York Times, has made several references to people who 'do not exist'. Your job is to wipe out the old, incorrect names and anonymize them, replacing them with the Ministry's preferred name, John Smith.

There are far too many documents for the number of employees in the Ministry of Truth to manually comb over and make the relevant adjustments. The Ministry would like to write a program to do this automatically.

## 3.1 Task (70 points)

Implement the method `clean` in the class `Cleaner` found in the file `cleaner.py`. A blank template has been provided for you.

The `clean` method should take in three arguments: a list of banned full names, a list of banned last names, and an input string to process. It should return as output the input string after the replacement of the banned full names and last names, if any. Specifically, the method should:

1. Replace all instances of the banned full names with the officially approved name `John Smith`
2. Replace all banned last names with the officially approved last name `Smith`

More specific details can be found in the docstring of the class and method.

## 3.2 Specific Example

Suppose we have a segment of text as follows, where the name `Tiger Woods` is on the banned list:

```
I believe that Tiger Woods is an amazing golfer. Tiger Woods' abilities are far beyond my own.
My favorite golfer is Tiger Woods, and I have a shirt with TIGER WOODS printed on it.
The article named Tiger-Woods was published yesterday. Woods are the material that form the
trunks and branches of trees.
```

After careful processing by your program, this segment should read:

```
I believe that John Smith is an amazing golfer. John Smith' abilities are far beyond my own.
My favorite golfer is John Smith, and I have a shirt with TIGER WOODS printed on it.
The article named Tiger-Woods was published yesterday. Smith are the material that form the
trunks and branches of trees.
```

## 3.3 Testing

You can test your program locally by running `cleaner.py` as an executable. Template code has been provided to read an input file and a file containing the banned names, one on each line.

You can run the executable as follows. This will read in the text data from `input.txt` and print the processed output to STDOUT.

```
$ python cleaner.py banned_names.txt input.txt
```

## 3.4 Grading

Only your implementation of the `clean` method in the `Cleaner` class will be graded. The executable part of the script which you used for testing will not be run (so feel free to modify those segments for your needs during testing).

Your implementation will be run against a variety of test cases that will look at normal and edge case behavior of your code (70 points). It it thus important that you implement specifically what is necessary, not more or less.

There is no style grading.

We will also run your implementation on the 1000 text files and compare the output against our gold standard. This score will then be visible on the leaderboard. This is not part of your grade (it may be possible to pass all the test cases but not get 100% for the leaderboard).

## 3.5 Frequently Asked Questions

We often receive questions regarding what exactly is an instance of a full name and last name. The rule of thumb is that an instance of a full name always consists of two words, separated only by some whitespace. Each word must be properly capitalized (first letter of each word should be capitalized, the rest should be lower caps) for it to be a full name. Similarly, an instance of a last name is always a single word that is properly capitalized.

Other common questions:

1. Other libraries - you should not need to use any libraries other than the standard Python libraries
2. Regex vs other methods - please implement a Regex based solution. The autograder will accept other solution designs without penalty, however we will not provide support for non-Regex based solutions at office hours.
3. Whitespace - your solution should preserve the original whitespace in the input source, if any. Do not add, remove, or replace any whitespace.
4. Punctuation - your implementation will be tested against inputs that contain common punctuation. The NYT data set contains numerous examples of replacement contexts containing punctuation. You can use those examples to test your implementation.
5. Can a last name be a first name? - Yes, but you should prioritize full name replacement over last name replacement if possible.
6. Is that (e.g. Gray) a last name or something else? - You have no way of knowing at this point, so if it looks like a last name, it is a last name. Replace it.

# 4 Part 2: Decoding Messages (30 points)

The Ministry of Truth has intercepted a lengthy encrypted message from a subversive organization known by the codename `Mammals of Insufferable Temperament` (MIT). Because MIT lacks knowledge of modern cryptographic methods, they use simple substitution ciphers to encrypt messages. Those working for MIT speak an inelegant (but understandable) dialect of English. We have intercepted a message from MIT, the contents of which can be found in the file `encoded.txt`.

## 4.1 Task (30 points)

Your task is to figure out the specific cipher used to create this encoded MIT document `encoded.txt`.

Specifically, you will submit a key file `key.txt`, where each line contains two alphabet letters separated by a space. Substitution is performed on all 26 letters of the alphabet, for both capital letters and non-capital letters, so there should be a total of 52 lines in your key file. A useless but submittable key file has been provided for you as a template.

The key file maps from the first column to the second column of letters. i.e. given the original plain text document, substituting letters from the first column with the respective letters from the second column will produce the encoded file. Naturally, given the encoded file `encoded.txt`, substituting letters from the second column with the respective letters from the first column will recover the original plain text document.

To accomplish this task without brute forcing, you should make use of some heuristics about the properties of language to help narrow your search space. Specifically, you can assume that the encoded document is sampled from the same distribution as the NYT data set, from which you can make some assumptions about the relative distribution of words and letters (we would expect it to be similar).

A suggested workflow is as follows.

1. Perform frequency analysis on the data from Part 1 and the encrypted message
2. Improve key for some letters heuristically based on the frequency analysis
3. Attempt to decrypt the message using current mapping
4. Look at the output and manually refine the mappings, repeat step 2 to 4

## 4.2 Testing

To help you with this process, you can test your key locally by running `decode.py` with your key file as follows. Feel free to modify the file for your needs.

```
$ python decode.py decode encoded.txt key.txt
```

## 4.3 Grading

You will be graded on the proportion of correctly mapped letters in your key file `key.txt`.

# 5 Submission

You should submit the following files as a compressed zip file on Gradescope. Please ensure that these files are available at the top level of your compressed file directory, and not nested in a parent folder. Alternatively you can just drag and drop these files into Gradescope's submission interface.

1. `cleaner.py` - implementation for Part 1
2. `key.txt` - decoding key for Part 2
3. `README` - (optional)

Your code should not take too long to run (2 minutes or less), unless you have an extremely inefficient implementation. You should get your score immediately afterwards. If you have failed some test cases, the grading interface should reflect the specific test case you failed and provide you a description of that test case so you can fix your implementation.

If you have consulted with other students or resources, you must indicate it in the `README` file.