Assignment 1:
The Noisy Channel and a Probabilistic Spell Checker

Submission deadline 23:59, April 30 (Sunday, just before midnight)

In this assignment you'll build a spell checker that handles both non-word and real-word errors given in a sentential context. In order to do that you will have to learn a language model and use error matrices, combining it all to a context sensitive noisy channel model.

The purpose of this assignment is to give you hands-on experience with probabilistic models of language by implementing the full algorithmic pipeline.

Specifics:

- You should implement a class called `Spell_Checker` (API below)
- Use the noisy channel model to correct the errors, that is - a correction of a misspelled word depends on both the most probable correction on the **error type-character level** and words prior; A correction of a word in a sentence depends on the **error type-character level and on the language model** -- the correction should maximize the likelihood of getting the full corrected sentence.
- Use the language model when correcting words in a sentential context.
- You should assume a word has at most two errors (that is, 'character' will be considered as a correction for 'karacter' [sub+del], while it will not be considered for 'karakter').
- You should assume a sentence has at most one erroneous word.
- You have to implement the API specified below

API

Your code should work seamlessly with the following API:

```
def __init__(self, lm=None)
def add_language_model(self, lm)
def add_error_tables(self, error_tables)
def spell_check(self, text, alpha)
def evaluate_text(self,text)
def who_am_i()  #not a class method
class Language_Model
```

The inner class `Language_Model` should support the following API:

```
def __init__(self, n=3, chars=False)
def build_model(self, text)
def get_model_dictionary(self)
def get_model_window_size(self)
def generate(self, context=None, n=20)
def evaluate_text(self,text)
def smooth(self, ngram)

def normalize_text(text)   #this is a global function
```

See detailed specifications and documentation here.

Corpora

Here are some corpora to use:

- Norvig's big.txt file (make sure to look at the file and its format)
- An even bigger corpus (preprocessed, sentences are separated by <s>)
- Trump's historical tweets (~14K tweets and retweets by Trump, each tweet in a new line)

Feel free (=you are encouraged) to experiment with other resources (e.g. nltk.corpus)

***Think**: make sure to understand the impact of the different corpora.*

Error Lists

- A file containing the confusion matrices used in A Spelling Correction Program Based on a Noisy Channel Model, Kernighan, Church and Gale, 1990: error tables
- Extra: Create your error types matrices from this list of common errors (containing only the single error pairs in Wikipedia common misspellings). File format: each line is a tab separated tuple: `<error>  <correct>`

***Think**:*

1. *What are the drawbacks of using the common_errors file to learn the confusion matrices?*
2. *Are there any significant performance differences between using the confusion matrices learned from the common errors file and using the given error tables?*

Efficiency

While this course is not about software engineering and code design, your code is expected to be reasonably efficient. The efficiency will not be evaluated or scored, but if the time it takes to create the language model or correct a sentence is too long (=minutes) we will stop the execution and your submission will not be checked.
However, we recommend having a working code first, then improve running time if necessary.

Submission Guidelines

1. You should use the course Moodle to submit a **single gz** file containing a single python file. (and only gz!)
2. Your code file should be named **ex1.py**, and must contain all necessary methods/functions/classes that support the specified API.
3. Your code **shouldn't print anything** to standard output!
4. You should use **python 3.9**.
5. You can **only** import the following modules: re, sys, random, math, collections, nltk
6. You should **document your code** using either Google Style or Pyhton PEP 257.

Sandbox

We recommend you play with your code a bit - using different corpora to generate the language model, using different n for the n-grams and and play with the various parameters that govern the language model and language generation, and sensitivity to errors.

Integrity and Cooperation

You should work on your assignments alone and refrain from sharing code snippets. However, you are encouraged to discuss various aspects of the assignment in the dedicated assignment forum and you are welcome to share testers and additional corpora.
After you submit your solutions may sample a small number of students for a short interview to discuss their implementation decisions.

Tips

1. Check out Norvig's spell checker for some coding tricks.
2. Don't rush into coding - plan ahead! Think about the different modules you will need to implement.
3. Consider adding instance variables and methods for efficiency and clarity. For example, another representation of the language model. Here is an example: Markov_LM_toy.ipynb (was supposed to be ex0)