Assignment 1

This assignment will involve the creation of a spellchecking system and an evaluation of its performance. You may use the code snippets provided in Python for completing this or you may use the programming language or environment of your choice

Please start by downloading the corpus `holbrook.txt` from Blackboard

The file consists of lines of text, with one sentence per line. Errors in the line are marked with a `|` as follows

```
My siter|sister go|goes to Tonbury .
```

In this case the word 'siter' was corrected to 'sister' and the word 'go' was corrected to 'goes'.

In some places in the corpus two words maybe corrected to a single word or one word to a multiple words. This is denoted in the data using underscores e.g.,

```
My Mum goes out some_times|sometimes .
```

For the purpose of this assignment you do not need to separate these words, but instead you may treat them like a single token.

Note: you may use any functions from NLTK to complete the assignment. It should not be necessary to use other libraries and so please consult with us if your solution involves any other external library. If you use any function from NLTK in Task 6 please include a brief description of this function and how it contributes to your solution.

Task 1 (10 Marks)

Write a parser that can read all the lines of the file `holbrook.txt` and print out for each line the original (misspelled) text, the corrected text and the indexes of any changes. The indexes refers to the index of the words in the sentence. In the example given, there is only an error in the 10th word and so the list of indexes is [9]. It is not necessary to analyze where the error occurs inside the word.

Then split your data into a test set of 100 lines and a training set.

```
lines = open("holbrook.txt").readlines()
data = []
# Write your code here

#assert(data[2] == {
#    'original': ['I', 'have', 'four', 'in', 'my', 'Family', 'Dad', 'Mum', 'and',
#    'corrected': ['I', 'have', 'four', 'in', 'my', 'Family', 'Dad', 'Mum', 'and',
#    'indexes': [9]
#})
```

The counts and assertions given in the following sections are based on splitting the training and test set as follows

```
test = data[:100]
train = data[100:]
```

Task 2 (10 Marks):

Calculate the frequency (number of occurrences), *ignoring case*, of all words and their unigram probability from the corrected *training* sentences.

Hint: use `Counter` to implement this so it may be called many times

```
from collections import Counter

def unigram(word):
    # Write your code here.
    return 0


def prob(word):
    # Write your code here.
    return 0

# Test your code with the following
#assert(unigram("me")==87)
```

Task 3 (15 Marks):

Edit distance is a method that calculates how similar two strings are to one another by counting the minimum number of operations required to transform one string into the other. There is a built-in implementation in NLTK that works as follows:

```
from nltk.metrics.distance import edit_distance

# Edit distance returns the number of changes to transform one word to another
print(edit_distance("hello", "hi"))
```

Write a function that calculates all words with *minimal* edit distance to the misspelled word. You should do this as follows

1. Collect the set of all unique tokens in `train`
2. Find the minimal edit distance, that is the lowest value for the function `edit_distance` between `token` and a word in `train`
3. Output all unique words in `train` that have this same (minimal) `edit_distance` value

Do not implement edit distance, use the built-in NLTK function `edit_distance`

```
def get_candidates(token):
    # Write your code here.
    return []

# Test your code as follows
#assert get_candidates("minde") == ['mine', 'mind']
```

Task 4 (15 Marks):

Write a function that takes a (misspelled) sentence and returns the corrected version of that sentence. The system should scan the sentence for words that are not in the dictionary (set of unique words in the training set) and for each word that is not in the dictionary choose a word in the dictionary that has minimal edit distance and has the highest *unigram probability*.

Your solution to this should involve get_candidates

```python
def correct(sentence):
    # Write your code here
    return sentence

#assert(correct(["this","whitr","cat"]) = ['this','white','cat'])
```

Task 5 (10 Marks):

Using the test corpus evaluate the *accuracy* of your method, i.e., how many words from your system's output match the corrected sentence (you should count words that are already spelled correctly and not changed by the system).

```python
def accuracy(test):
    # Write your code here
    return 0.0

print(accuracy(test))
```

Task 6 (35 Marks):

Consider a modification to your algorithm that would improve the accuracy of the algorithm developed in Task 3 and 4

- You may resources beyond those provided here.
- You must **not use the test data** in this task.
- Provide a short text describing what you intend to do and why.
- Full marks for this section may be obtained without an implementation, but an implementation is preferred.
- Your implementation should not consist of more than 50 lines of code

Please note this task is marked according to: demonstration of knowledge from the lecutures (10), originality and appropriateness of solution (10), completeness of description (10) and technical correctness (5)

Task 7 (5 Marks):

Repeat the evaluation (as in Task 5) of your new algorithm and show that it outperforms the algorithm from Task 3 and 4