A Lab Assignment Report Submitted for
Natural Language Processing(UML 602)

On

SPELL CHECKER

Submitted By :

Keshav Gupta - 101603160
Kunal Kumar - 101603168

Submitted to -
Ms. Diksha Hooda

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department of Computer Science and Engineering
Thapar Institute of Engineering and Technology
Patiala, Punjab - 147001
Jan - May 2019

Table of Contents

1. Basic Introduction to NLP:

Natural language processing refers to the use and ability of systems to process sentences in a natural language such as English, rather than in specialized artificial computer languages such as C, C++, JAVA, PROLOG etc.

Importance of communication with computers in everyday language cannot be overlooked just imagine if we give a simple command to computer like "open Microsoft word" and computer behaves accordingly, life will be much simplified. A natural language processing system can be defined as a system that performs useful operations on natural language inputs. Research on NLP began in the 1950's.

Presently, in the 21st century, NLP research and development is booming. An indication of the development of natural language processing systems is that they are increasingly being used in support of other computer programs.

2. Introduction to Spell Checker Application :

According to Wikipedia, Spell Checker is mainly a software feature which is generally used to check whether the words in particular file are correctly spelled or not.

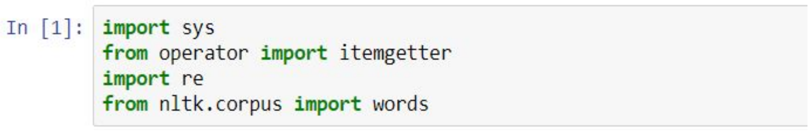
A basic spell checker usually carries out the following processes:

- Firstly it scans the file and separate out words from it.
- After scanning, it generally compares the extracted word with the dictionary of correctly spelled words and based upon certain measures such as min edit distance and morphological analysis, it predicts whether the word is correct or not.
- Once the word is found to be misspelled then it shows the user the possible combinations with which that word could be replaced.

3. Working of Code :

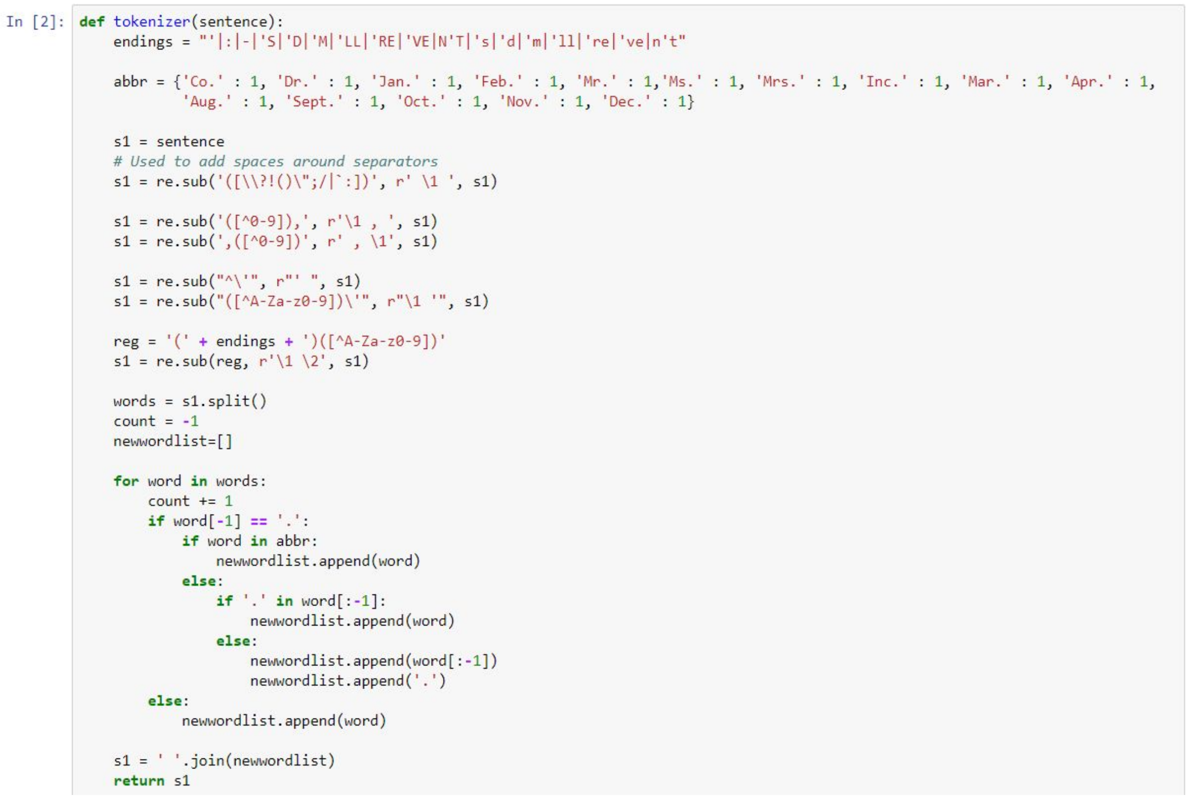
During the implementation of code of Spell Checker application, modularised approach is being used so as to improve the code readability as well as implementation.
In this working, different modules and methods are being shown here along with their code as well as explanation.

- Firstly required python libraries such as NLTK, OS, etc. are being installed as shown below:



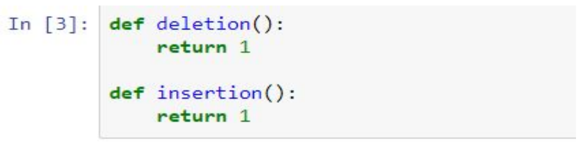
```
In [1]: import sys
        from operator import itemgetter
        import re
        from nltk.corpus import words
```

- **Tokenisation :** Tokenisation is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some characters like punctuation marks are discarded. The tokens become the input for another process like parsing and text mining.



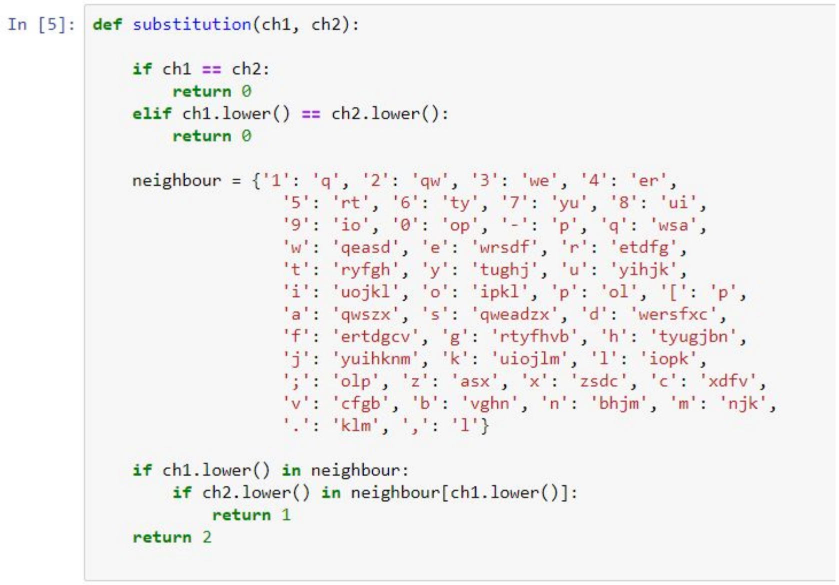
```
In [2]: def tokenizer(sentence):
            endings = "'|:|-|'S|'D|'M|'LL|'RE|'VE|N'T|'s|'d|'m|'ll|'re|'ve|n't"

            abbr = {'Co.' : 1, 'Dr.' : 1, 'Jan.' : 1, 'Feb.' : 1, 'Mr.' : 1,'Ms.' : 1, 'Mrs.' : 1, 'Inc.' : 1, 'Mar.' : 1, 'Apr.' : 1,
                    'Aug.' : 1, 'Sept.' : 1, 'Oct.' : 1, 'Nov.' : 1, 'Dec.' : 1}

            s1 = sentence
            # Used to add spaces around separators
            s1 = re.sub('([\\?!()\";/|`:])', r' \1 ', s1)

            s1 = re.sub('([^0-9]),', r'\1 , ', s1)
            s1 = re.sub(',([^0-9])', r' , \1', s1)

            s1 = re.sub("^\'", r"' ", s1)
            s1 = re.sub("([^A-Za-z0-9])\'", r"\1 '", s1)

            reg = '(' + endings + ')([^A-Za-z0-9])'
            s1 = re.sub(reg, r'\1 \2', s1)

            words = s1.split()
            count = -1
            newwordlist=[]

            for word in words:
                count += 1
                if word[-1] == '.':
                    if word in abbr:
                        newwordlist.append(word)
                    else:
                        if '.' in word[:-1]:
                            newwordlist.append(word)
                        else:
                            newwordlist.append(word[:-1])
                            newwordlist.append('.')
                else:
                    newwordlist.append(word)

            s1 = ' '.join(newwordlist)
            return s1
```

- **Deletion and Insertion of Character:** This method is made so as to get the cost of deletion and insertion of character at the time of prediction of correct word from a misspelled word.



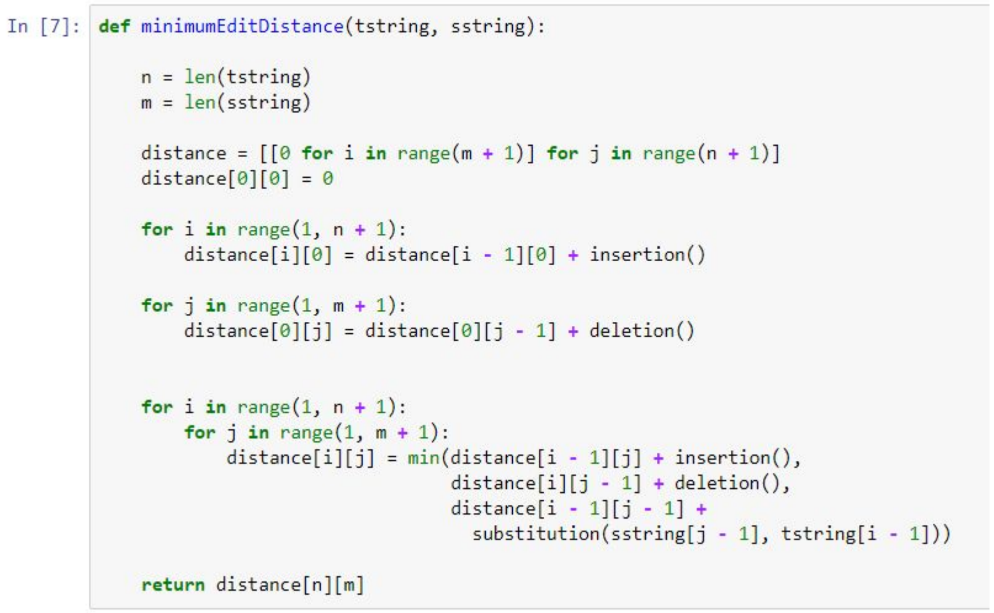
```
In [3]: def deletion():
            return 1

        def insertion():
            return 1
```

- **Substitution :** This method is made in order to compute the cost for replacing one character with another.



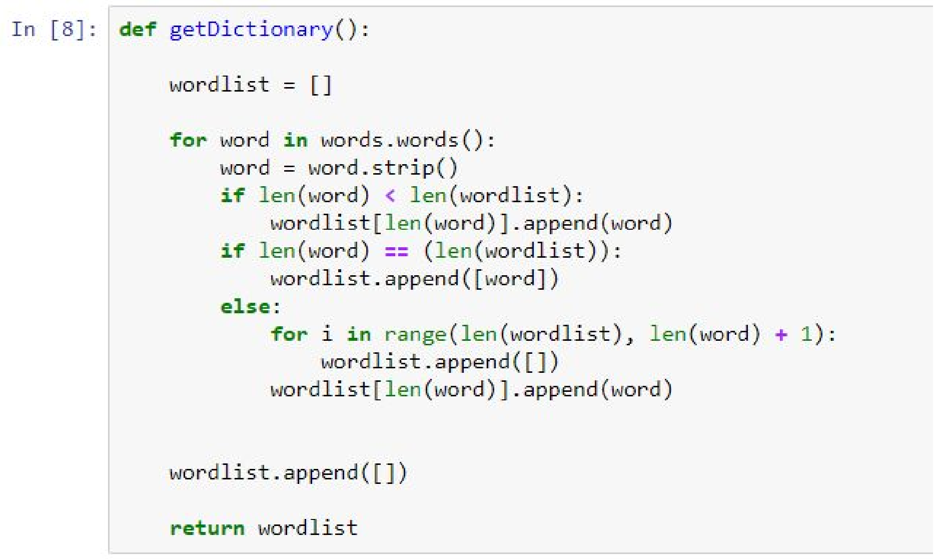
```
In [5]: def substitution(ch1, ch2):

            if ch1 == ch2:
                return 0
            elif ch1.lower() == ch2.lower():
                return 0

            neighbour = {'1': 'q', '2': 'qw', '3': 'we', '4': 'er',
                         '5': 'rt', '6': 'ty', '7': 'yu', '8': 'ui',
                         '9': 'io', '0': 'op', '-': 'p', 'q': 'wsa',
                         'w': 'qeasd', 'e': 'wrsdf', 'r': 'etdfg',
                         't': 'ryfgh', 'y': 'tughj', 'u': 'yihjk',
                         'i': 'uojkl', 'o': 'ipkl', 'p': 'ol', '[': 'p',
                         'a': 'qwszx', 's': 'qweadzx', 'd': 'wersfxc',
                         'f': 'ertdgcv', 'g': 'rtyfhvb', 'h': 'tyugjbn',
                         'j': 'yuihknm', 'k': 'uiojlm', 'l': 'iopk',
                         ';': 'olp', 'z': 'asx', 'x': 'zsdc', 'c': 'xdfv',
                         'v': 'cfgb', 'b': 'vghn', 'n': 'bhjm', 'm': 'njk',
                         '.': 'klm', ',': 'l'}

            if ch1.lower() in neighbour:
                if ch2.lower() in neighbour[ch1.lower()]:
                    return 1
            return 2
```

- **MinEditDistance :** This method computes the minimum cost in order to transform one string into another. In this method, DP approach is used to calculate the minimum distance.



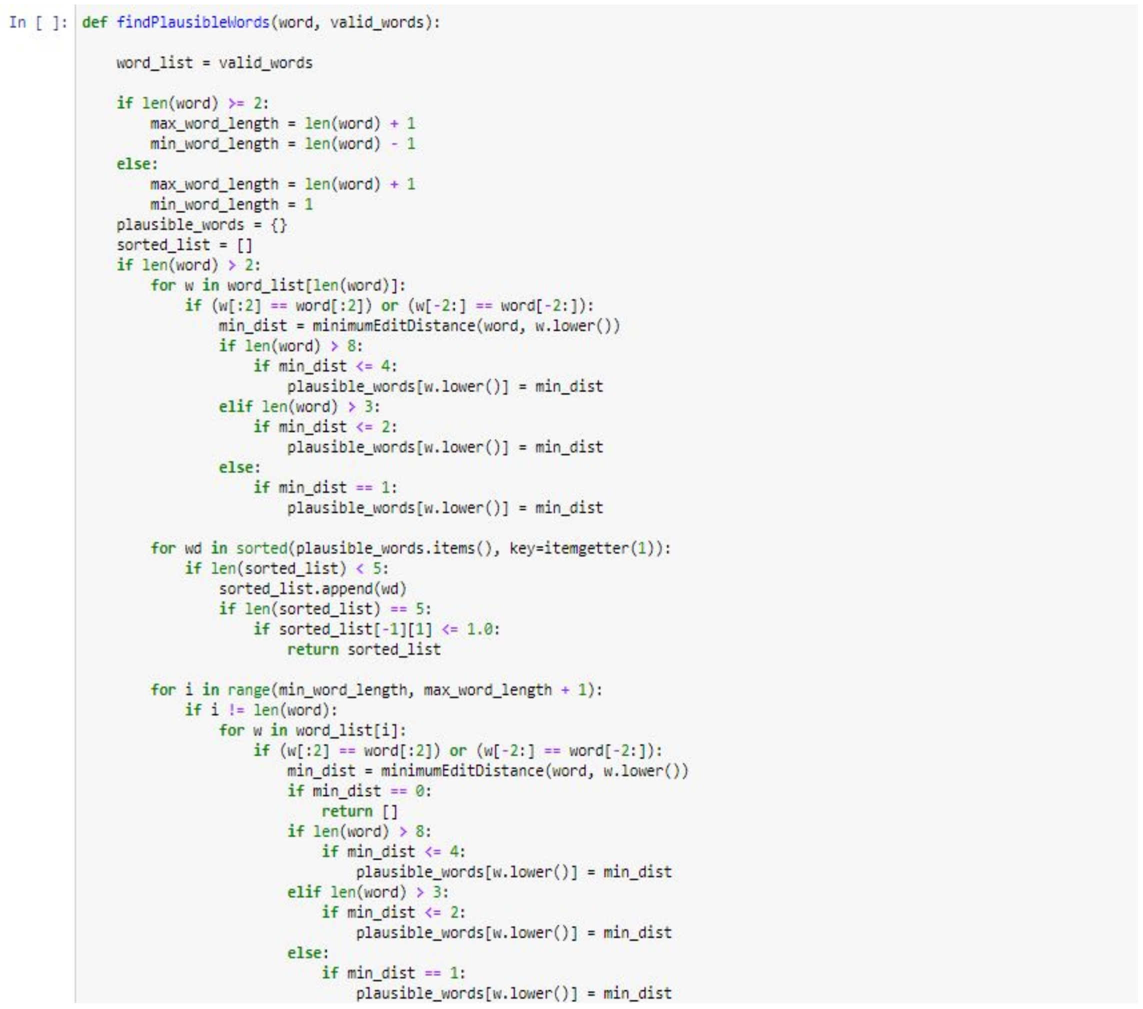
```
In [7]: def minimumEditDistance(tstring, sstring):

            n = len(tstring)
            m = len(sstring)

            distance = [[0 for i in range(m + 1)] for j in range(n + 1)]
            distance[0][0] = 0

            for i in range(1, n + 1):
                distance[i][0] = distance[i - 1][0] + insertion()

            for j in range(1, m + 1):
                distance[0][j] = distance[0][j - 1] + deletion()


            for i in range(1, n + 1):
                for j in range(1, m + 1):
                    distance[i][j] = min(distance[i - 1][j] + insertion(),
                                         distance[i][j - 1] + deletion(),
                                         distance[i - 1][j - 1] +
                                           substitution(sstring[j - 1], tstring[i - 1]))

            return distance[n][m]
```

- **GetDictionary :** This method sorts the dictionary of words from nltk.corpus on the basis of their size and returns a separate list.

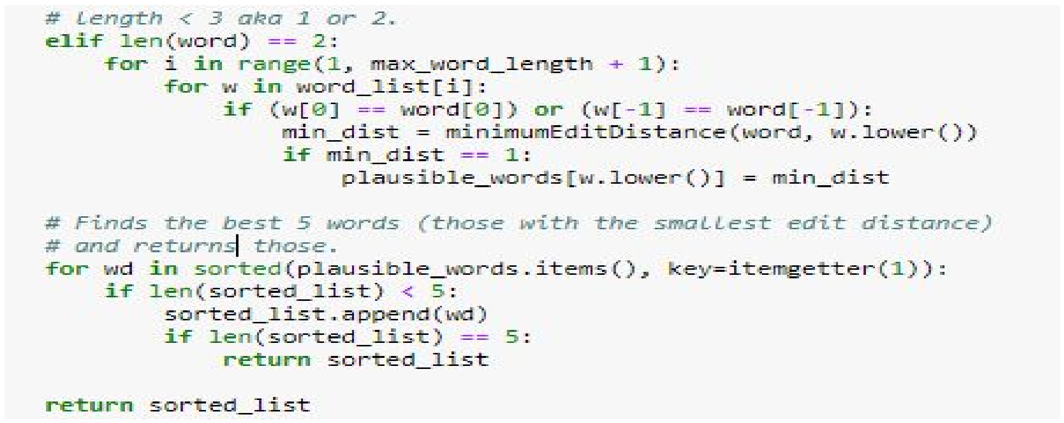


```
In [8]: def getDictionary():

            wordlist = []

            for word in words.words():
                word = word.strip()
                if len(word) < len(wordlist):
                    wordlist[len(word)].append(word)
                if len(word) == (len(wordlist)):
                    wordlist.append([word])
                else:
                    for i in range(len(wordlist), len(word) + 1):
                        wordlist.append([])
                    wordlist[len(word)].append(word)


            wordlist.append([])

            return wordlist
```

- **findPlausibleWords:** This method takes a word and a list of valid words and checks whether the word is valid or not. If it is not valid the minimum edit distance is calculated for a series of valid words and the ones with the lowest edit distance are returned.

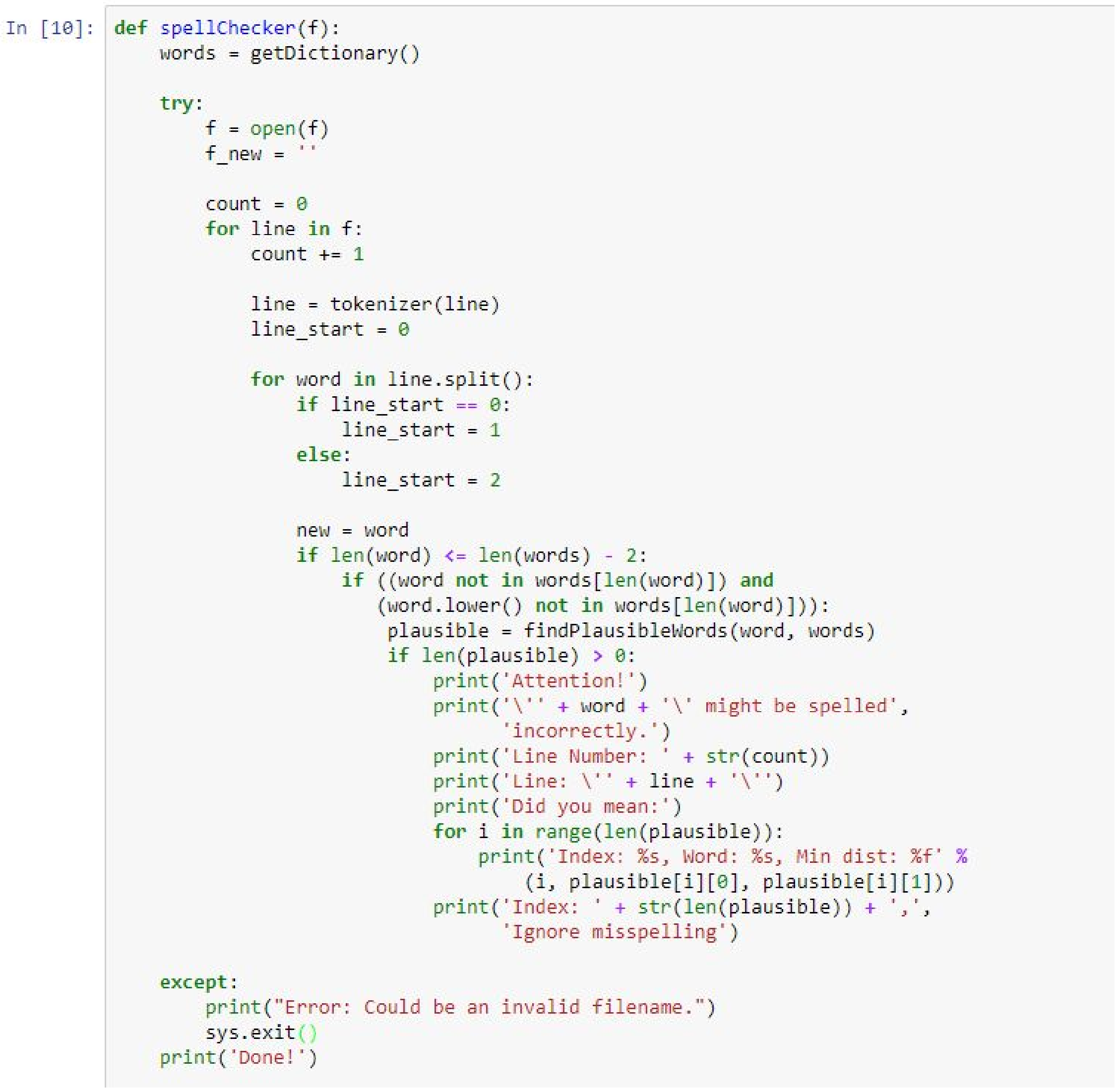


```
In [ ]: def findPlausibleWords(word, valid_words):

            word_list = valid_words

            if len(word) >= 2:
                max_word_length = len(word) + 1
                min_word_length = len(word) - 1
            else:
                max_word_length = len(word) + 1
                min_word_length = 1
            plausible_words = {}
            sorted_list = []
            if len(word) > 2:
                for w in word_list[len(word)]:
                    if (w[:2] == word[:2]) or (w[-2:] == word[-2:]):
                        min_dist = minimumEditDistance(word, w.lower())
                        if len(word) > 8:
                            if min_dist <= 4:
                                plausible_words[w.lower()] = min_dist
                        elif len(word) > 3:
                            if min_dist <= 2:
                                plausible_words[w.lower()] = min_dist
                        else:
                            if min_dist == 1:
                                plausible_words[w.lower()] = min_dist

                for wd in sorted(plausible_words.items(), key=itemgetter(1)):
                    if len(sorted_list) < 5:
                        sorted_list.append(wd)
                        if len(sorted_list) == 5:
                            if sorted_list[-1][1] <= 1.0:
                                return sorted_list

                for i in range(min_word_length, max_word_length + 1):
                    if i != len(word):
                        for w in word_list[i]:
                            if (w[:2] == word[:2]) or (w[-2:] == word[-2:]):
                                min_dist = minimumEditDistance(word, w.lower())
                                if min_dist == 0:
                                    return []
                                if len(word) > 8:
                                    if min_dist <= 4:
                                        plausible_words[w.lower()] = min_dist
                                elif len(word) > 3:
                                    if min_dist <= 2:
                                        plausible_words[w.lower()] = min_dist
                                else:
                                    if min_dist == 1:
                                        plausible_words[w.lower()] = min_dist
```



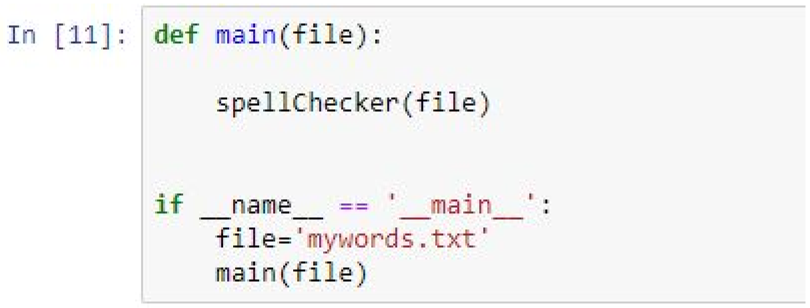
```
# Length < 3 aka 1 or 2.
elif len(word) == 2:
    for i in range(1, max_word_length + 1):
        for w in word_list[i]:
            if (w[0] == word[0]) or (w[-1] == word[-1]):
                min_dist = minimumEditDistance(word, w.lower())
                if min_dist == 1:
                    plausible_words[w.lower()] = min_dist

# Finds the best 5 words (those with the smallest edit distance)
# and returns those.
for wd in sorted(plausible_words.items(), key=itemgetter(1)):
    if len(sorted_list) < 5:
        sorted_list.append(wd)
        if len(sorted_list) == 5:
            return sorted_list

return sorted_list
```

- **SpellChecker :** This method takes a file from user and checks whether the words present in the file are correctly spelled or not according to the dictionary.



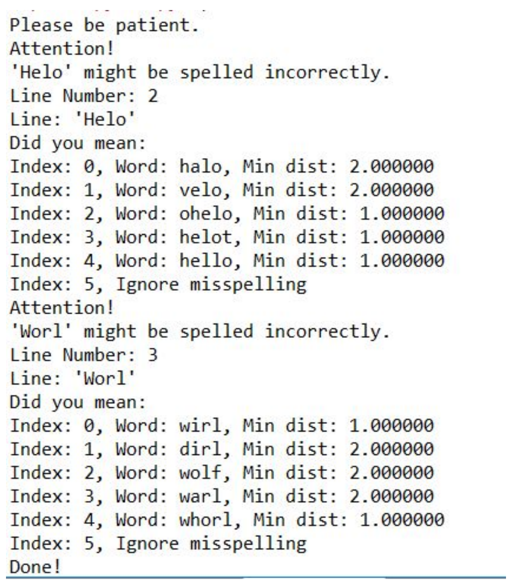
```
In [10]: def spellChecker(f):
             words = getDictionary()

             try:
                 f = open(f)
                 f_new = ''

                 count = 0
                 for line in f:
                     count += 1

                     line = tokenizer(line)
                     line_start = 0

                     for word in line.split():
                         if line_start == 0:
                             line_start = 1
                         else:
                             line_start = 2

                         new = word
                         if len(word) <= len(words) - 2:
                             if ((word not in words[len(word)]) and
                                (word.lower() not in words[len(word)])):
                                 plausible = findPlausibleWords(word, words)
                                 if len(plausible) > 0:
                                     print('Attention!')
                                     print('\'' + word + '\' might be spelled',
                                           'incorrectly.')
                                     print('Line Number: ' + str(count))
                                     print('Line: \'' + line + '\'')
                                     print('Did you mean:')
                                     for i in range(len(plausible)):
                                         print('Index: %s, Word: %s, Min dist: %f' %
                                               (i, plausible[i][0], plausible[i][1]))
                                     print('Index: ' + str(len(plausible)) + ',',
                                           'Ignore misspelling')

             except:
                 print("Error: Could be an invalid filename.")
                 sys.exit()
             print('Done!')
```

- **Main :** This method starts the execution of our python program in which the respective file is entered which is to be checked.



```
In [11]: def main(file):

             spellChecker(file)


         if __name__ == '__main__':
             file='mywords.txt'
             main(file)
```

Output of Code:



```
Please be patient.
Attention!
'Helo' might be spelled incorrectly.
Line Number: 2
Line: 'Helo'
Did you mean:
Index: 0, Word: halo, Min dist: 2.000000
Index: 1, Word: velo, Min dist: 2.000000
Index: 2, Word: ohelo, Min dist: 1.000000
Index: 3, Word: helot, Min dist: 1.000000
Index: 4, Word: hello, Min dist: 1.000000
Index: 5, Ignore misspelling
Attention!
'Worl' might be spelled incorrectly.
Line Number: 3
Line: 'Worl'
Did you mean:
Index: 0, Word: wirl, Min dist: 1.000000
Index: 1, Word: dirl, Min dist: 2.000000
Index: 2, Word: wolf, Min dist: 2.000000
Index: 3, Word: warl, Min dist: 2.000000
Index: 4, Word: whorl, Min dist: 1.000000
Index: 5, Ignore misspelling
Done!
```

4. Other Applications of NLP :

- Text Classification and Categorization.
- POS Tagging
- Speech Recognition
- Question Answering Systems