Project

Spell Checker

what’s a spell checker?

How to..

Build a spell checker for dummies ?

- We’ll start with a simple version and build on it.

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

Get inputs

For this software, it needs two inputs:

1. A word to search for.
2. A dictionary list to search in.

```
word=input("type a word ")

word.lower()
```

Get inputs

For this software, it needs two inputs:

1. A word to search for.
2. A dictionary list to search in.

```
def read_dict(dict_name):
    """"reads words from txt file into list"""
    text_file = open(dict_name, "r")
    dict=text_file.read().splitlines()
    text_file.close()
    return dict
```

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

Search for the word

Binary search can be coded in two methods, recursion or iteration.

It has simple unique format

```
def find(dict, word):
   l = 0
   r = len(dict) - 1
   while r>=l:
       mid=(r+l)//2
       if dict[mid]==word:
           return True
       elif..
```

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

show the user some options

For this simple version, we only need 3 choices which are the closest matches to the input in the dictionary.

There’s is a function called get_close_matches() in the difflib module that does that job for you

Show options

get_close_matches() is based on a function called SequaceMatcher()
It returns a list with maximum 3 elements by default

Super easy, but we want to know how it works under the hood

```
replace=difflib.get_close_matc/h
es(word,dict,3)
```

Show options

SequaceMatcher turns any sequence of chars to a number using ASCII
It compares the numbers and returns a ratio of difference

Set_seq : computes and caches detailed information about the sequence

Ratio: Return a measure of the sequences' similarity as a float in the range [0, 1]

```
from difflib import SequenceMatcher

def matches(word, possibilities, n=3,
cutoff=0.6):

    result = []

    s = SequenceMatcher()

    s.set_seq2(word)

    for x in possibilities:

        s.set_seq1(x)

        if s.real_quick_ratio() >=
cutoff and...
```

Show options

Now since we have the replacements ready in a list, we'll print them out and receive the user's choice

We want to make sure to get correct input, so we'll make an infinite loop that breaks only on having a correct input

```
while True:

    if len(replace)<1:
#the list replace is empty, no close
matches found

        print("no close matches.")

        break

    else:

        print("did you mean:")

 …. .
```

1. Get inputs.
2. Search for the word.
3. If it’s there, Great!
4. else, show the user some options.
5. Update it..

Update it

This software we just made is actually useless in the real world,

1. No one wants to check one word, it’ll be always a phrase.
2. Not every word in the world is in the dictionary, user should be able to add to it.
3. User should be able to skip or modify the word during the execution.
4. It should be able to skip punctuation.

Updates

1. Type a phrase
2. Add to dictionary
3. Skip & modify
4. Skip punctuation

Type a phrase

We'll just use .split() method to split the phrase into words

Then iterate over them with for loop

```
phrase = input("put your text here ")
words = phrase.split()


for word in words:

    ...
```

Updates

1. Type a phrase
2. Add to dictionary
3. Skip & modify
4. Skip punctuation

Add to dictionary

Because we want to keep the dictionary from changing words by mistake, we'll make another txt file called p.dic (personal dictionary).

We want to add to word in its correct position in the sorted dictionary so that it would remain searchable, there's a function called insort() in bisect module that does that job.

Let's see how it works.

```
def p_dic(nw):

    """"adds a word to the personal
dictionary file(p_dic.txt) """

    print(nw, "is added")

    dict=read_dict("p_dic.txt")

    insort(dict, nw)

    f= open("p_dic.txt","w")

    for word in dict:

        f.write("%s\n" % word)

    f.close
```

Add to dictionary

It’s basically binary search, expect we are sure that the element we are looking for isn’t in the list

The search will break after it reaches the closest element to the one we are looking for, and that where you want to put it

```
def insort(a, x)
        hi = len(a)
        lo=0
    while lo < hi:
        mid = (lo+hi)//2
        if x < a[mid]: hi = mid
        else: lo = mid+1
    a.insert(lo, x)
```

Updates

1. Type a phrase
2. Add to dictionary
3. Skip & modify
4. Skip punctuation

Skip & modify

We'll just add a couple of lines to the action function

```
def action(dict, word, words, count):

    replace= matches(word,dict,3)
    choice=0

        while True:

            if len(replace)<1:
print("no close matches.")

...
```

Updates

1. Type a phrase
2. Add to dictionary
3. Skip & modify
4. Skip punctuation

Skip punctuation

We'll just add a simple function of ifs

```
if word[-1]=="."or word[-1]==","or
word[-1]=='"'or word[-1]==')'or
word[-1]==":":

    word= word[:-1]

if word[-3:]=="n't" or
word[-3:]=="'ve" or word[-3:]=="'re":

    word=word[:-3]

if word[-2:]=="'s" or word[-2:]=="'d":

    word=word[:-2]

if word[0]=='('or word[0]=='"':

    word=word[1:]
```

Thank you