

NLP HOLIDAY ASSIGNMENT

P.Janaki

2211cs020388

AIML-THETA

1] Correct the Search Query

<https://www.hackerrank.com/challenges/correct-the-search-query/problem?isFullScreen=true>

The screenshot displays the Hackerrank interface for the 'Correct the Search Query' challenge. On the left, the problem description states: 'You are provided with a search query where one or many words may be misspelled. Your task is to identify and fix the incorrectly spelled words in the query.' It provides an example where 'going to chiev' is corrected to 'going to chiev'. The input format specifies that the first line contains an integer N , the number of queries, and the next N lines contain search queries. The output format requires the output to contain N lines, each with the rectified search query. The code editor on the right contains a Python solution that uses a dictionary of words and the Levenshtein distance to find the closest match for each word in the query.

```
1 # For each word, find the closest match from the dictionary
2 closest_match = max(word_list, key=lambda dict_word: levenshtein_distance(word,
3 dict_word))
4 corrected_words.append(closest_match)
5
6 return " ".join(corrected_words)
7
8 def main():
9     # Read the number of queries
10    n = int(input().strip())
11
12    # Process each query
13    for _ in range(n):
14        query = input().strip()
15        corrected_query = correct_spelling(query)
16        print(corrected_query)
17
18 if __name__ == "__main__":
19     main()
20
```

The bottom of the page shows a green 'Congratulations' banner with the text 'You solved the challenge. Would you like to challenge your friends?' and a 'Next Challenge' button.

2] Deterministic Url and HashTag Segmentation

<https://www.hackerrank.com/challenges/url-hashtag-segmentation/problem?isFullScreen=true>

The screenshot shows the HackerRank interface for the challenge 'Deterministic Url and HashTag Segmentation'. The left sidebar contains the problem description, which explains that the task is to segment domain names and hashtags into tokens based on a list of 1000 most common words. The right pane displays a Python solution. The code defines a `tokenize` function that uses a dictionary of common words to split a string into tokens. It then processes the input string by splitting it at the boundaries of these tokens. The bottom of the interface shows a 'Run Code' button and a 'Submit Code' button, with a 'SUCCESS' message indicating the solution passed the tests.

```
14 def tokenize(input_string, dictionary):
15     return True
16     except ValueError:
17         return False
18
19 def tokenize(input_string, dictionary):
20     """
21     Tokenize the input string using the longest match first approach.
22     Args:
23         input_string: The string to be tokenized.
24         dictionary: A set of valid words.
25     Returns:
26         A list of tokens from the input string.
27     """
28     length = len(input_string)
29     if length == 0:
30         return []
31
32     # dp[i] stores the tokens for the substring ending at index i
33     dp = [None] * (length + 1)
34     dp[0] = [] # Base case: empty string has no tokens
```

3] Disambiguation: Mouse vs Mouse

<https://www.hackerrank.com/challenges/disambiguation-mouse-vs-mouse/problem?isFullScreen=true>

The screenshot shows the HackerRank interface for the challenge 'Disambiguation: Mouse vs Mouse'. The left sidebar contains the problem description, which states that the task is to identify if a sentence is about a computer mouse or an animal mouse based on the word 'mouse'. The right pane displays a Python solution. The code defines a `main` function that reads the number of test cases and processes each sentence. It checks for the presence of 'animal' or 'computer' keywords to disambiguate the word 'mouse'. The bottom of the interface shows a 'Run Code' button and a 'Submit Code' button, with a 'CONGRATULATIONS' message indicating the solution passed the tests.

```
11 if keyword in sentence:
12     return "computer mouse"
13
14 # Check for presence of animal mouse context
15 for keyword in animal_keywords:
16     if keyword in sentence:
17         return "animal"
18
19 # If no context is found, classify as animal (a default assumption)
20 return "animal"
21
22 # Main function to process input and output
23 def main():
24     # Read number of test cases
25     n = int(input())
26
27     # Process each sentence
28     for _ in range(n):
29         sentence = input().strip()
```

4] Language Detection

<https://www.hackerrank.com/challenges/language-detection/problem?isFullScreen=true>

Problem

Given a snippet of text in English, French, German, or Spanish, detect the snippet's language and print the language name. You may build an offline model for this. The snippet may contain one or more lines. You may make no more than 10 submissions for this problem, during the contest.

Input Format

One or more lines of text.

Constraints

- The snippet will not exceed 8 kilobytes in size.
- The snippet will be in one of the following languages: English, French, German, or Spanish.

Output Format

Print the snippet's language in Title Case, as written above, on a new line.

Sample Input

```
The story of Rip Van Winkle is set in the years before and after the American Revolution.
```

Sample Output

```
English
```

Explanation

The text snippet's language is English.

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
def detect_language(snippet):
    # Basic language-specific word patterns
    english_words = ("the", "and", "ing", "ed", "was", "with", "at", "of")
    french_words = ("le", "et", "est", "dans", "sur", "avec", "comme", "de")
    german_words = ("den", "und", "ist", "in", "auf", "mit", "als", "von")
    spanish_words = ("el", "y", "es", "en", "dentro", "con", "sobre", "de")

    # Tokenize the input snippet into words (case insensitive)
    words = snippet.lower().split()

    # Count overlaps with each language's word set
    english_overlap = len(words & english_words)
    french_overlap = len(words & french_words)
    german_overlap = len(words & german_words)
    spanish_overlap = len(words & spanish_words)

    # Find the language with the highest overlap
    overlaps = {
        "English": english_overlap,
        "French": french_overlap,
        "German": german_overlap,
        "Spanish": spanish_overlap
    }

    # Get the language with the maximum overlap
    detected_language = max(overlaps, key=overlaps.get)

    # If all overlaps are zero, return "Unknown"
    if all(value == 0 for value in overlaps.values()):
        return "Unknown"
```

5] The Missing Apostrophes

<https://www.hackerrank.com/challenges/the-missing-apostrophes/problem?isFullScreen=true>

Problem

Given a paragraph of text that has been stripped of any apostrophes, rewrite the paragraph with apostrophes (') inserted when appropriate. This is essentially the task of generative linguistics present in the text prediction engine of modern phones.

Dictionaries and Corpora of Text

We don't actually provide a particular dictionary or corpus or a set of features. To get started, you may find it useful to embed the list of 3000 common words as a dictionary in your program. For more effective representation models, you are encouraged to use your own word list, or corpora, or features extracted from a corpus, as required by whatever model you choose. *Project Gutenberg* is a good starting point, but keep in mind that language and its usage has evolved and transformed over time.

You may use *scikit-learn* to build and compare your model offline and to decompress and use it from your program. If you end up with a corpus or model that is too large, you may compress and serialize it. Then deserialize it from within your code using `dill` (that is in Python) or another tool. The reason that your model will contain a compressed string representing the dictionary which will then be decompressed and used. You can take a look at this code submitted during *CodeFest 2016*. For Java users, you might want to look up `java.util.zip.GZIPInputStream` for this purpose.

Input Format

A paragraph of text that is missing apostrophes single quoted. This paragraph may span multiple lines.

Scoring

All test cases are equally weighted. If A is the number of apostrophes missing from the text, C is the number of apostrophes you correctly insert back into the text, and W is the number of apostrophes you incorrectly insert into the text, your percentage score for the test will be $100 \times \max(\frac{C-A}{W}, 0)$.

If you add more apostrophes than is correct places, then the number of missing apostrophe marks you identify correctly, the score for the test case will be zero.

You may make no more than 10 submissions for this problem, during the contest.

Output Format

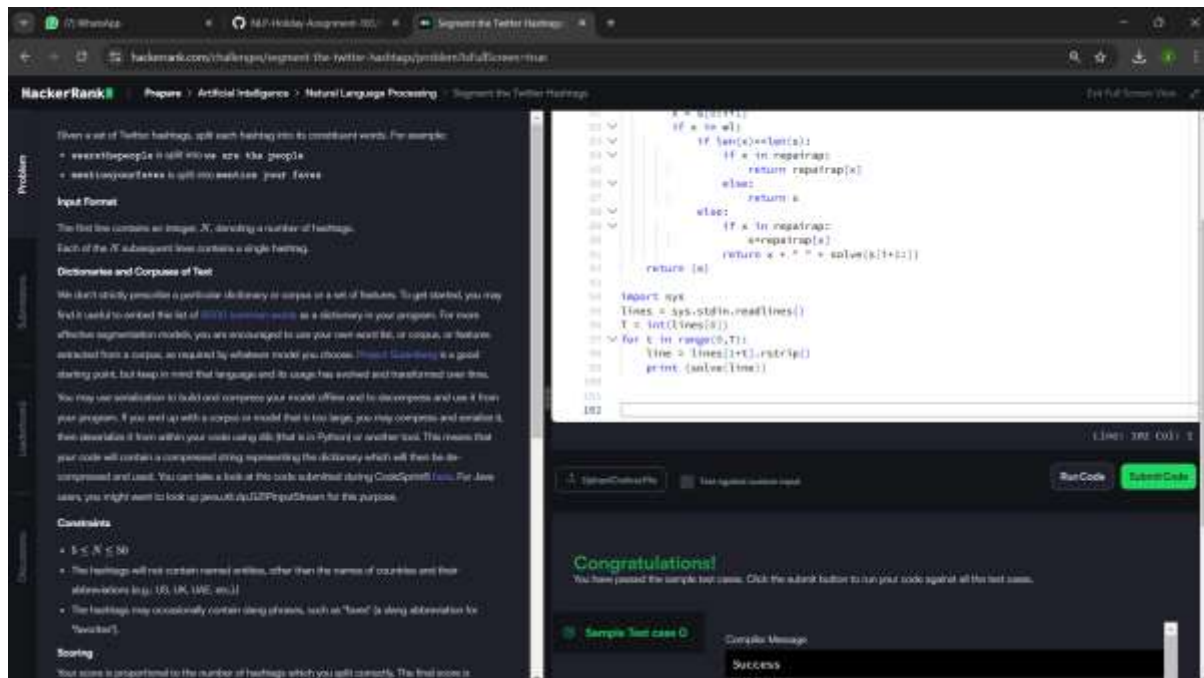
Print the corrected version of your input, apostrophes inserted where appropriate.

```
text = open('book4.txt').read().lower().replace("'", '')
text = open('book5.txt').read().lower().replace("'", '')
text = open('book6.txt').read().lower().replace("'", '')
text = "Atila's warriors' scarions' repellents' chemicals' w/e've' sensors' hormones' burn'
traels' culls' markers' censors' alerts' they'll' it's' proteins' indicators' parkinson's' lacka'
atropia's' pet's' beginnins' vaccines' tests' impacts' fava' researchers' material' vaccines' "
text = text.replace("a'", "is'").replace("were'", "is'").replace("parkinson's'", "parkinsons'")
words = list(set(re.findall("[a-z][a-z]*", text)))
words = " ".join(list(set(re.findall("[a-z][a-z]*", text))))
print("words: {}".format(base64.b64encode(zlib.compress(words.encode('utf-8'))).decode('utf-8')))
return base64.b64encode(zlib.compress(words.encode('utf-8'))).decode('utf-8'))

def getMissingWords():
    return b'e3cfd8a8qy78wvt+vg8T8p3g13ar9mvd0pqrw02ktr50m62FCUth0r3096/
cu4b6teVmd2a3454te7T072TMO3+7edufVw27886d8dU1M11306f23jSTsgTtyr/0GTS
+8r386862v3K1s87j70x2j59/0byHx8865rFHz0P92L2e13F0e4070y0M+Hv28P91L/aX70685
+H065W605+j9U08j1U7j7TV0377X0vdF06310b/u0zPv307+4vrt9623n58rbcd3Rv000n7w/
E5v83r406240588u05u7q3P/Xr+9Went6N9/kc7LwPv201k17/LxL0008k78j5+P993C3V
p6r+0L2py2f0e1PFFH044vnu/7r0mch0r3y9j3D54G077ed0Vr0r060r030d+2D5ca3D0w+td0uL
n6KtrPv0d2+uY9/zh0y9/112p/v0y9r06008r0h0k50M/REBTV00y06PvDUM24u081T0r+0h0Gtr
```

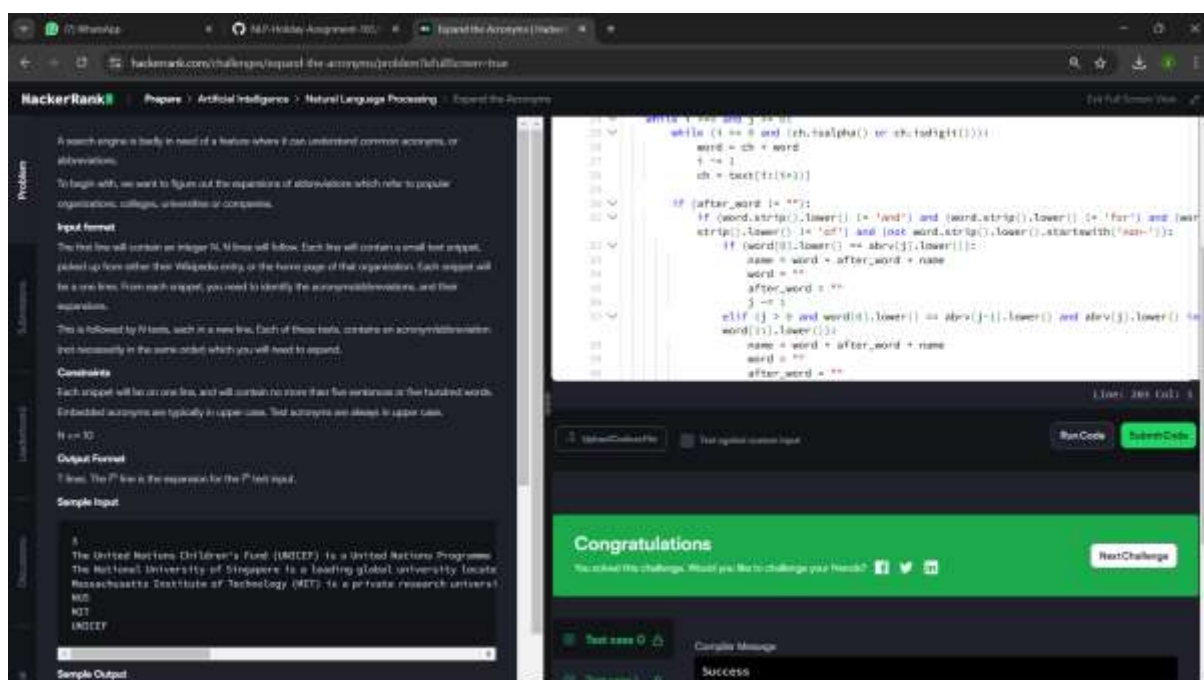
6] Segment the Twitter Hashtags

<https://www.hackerrank.com/challenges/segment-the-twitter-hashtags/problem?isFullScreen=true>



7] Expand the Acronyms

<https://www.hackerrank.com/challenges/expand-the-acronyms/problem?isFullScreen=true>



8] Correct the Search Query

<https://www.hackerrank.com/challenges/correct-the-search-query/problem?isFullScreen=true>

The screenshot shows the HackerRank interface for the 'Correct the Search Query' challenge. The left panel contains the problem description, which states that the task is to correct misspelled words in a search query. It provides an example where 'going to chier' is corrected to 'going to cheer'. The input format specifies that the first line contains the number of queries, and the next N lines contain the queries. The output format requires the corrected queries to be printed on separate lines. The right panel shows a Python solution that uses a dictionary of words and the Levenshtein distance to find the closest match for each word in the query. The solution includes a main function that reads the input, processes each query, and prints the corrected queries. Below the code editor, there is a 'Congratulations' message and a 'Next Challenge' button.

9] A Text-Processing Warmup

<https://www.hackerrank.com/challenges/a-text-processing-warmup/problem?isFullScreen=true>

The screenshot shows the HackerRank interface for the 'A Text-Processing Warmup' challenge. The left panel contains the problem description, which states that the task is to process a text fragment and identify the author and date. It provides a list of authors and their corresponding dates. The input format specifies that the first line contains the number of test cases, and the next N lines contain the test cases. The output format requires the author and date to be printed for each test case. The right panel shows a Python solution that uses a dictionary of authors and dates to identify the author and date for each test case. The solution includes a main function that reads the input, processes each test case, and prints the author and date. Below the code editor, there is a 'Congratulations' message and a 'Next Challenge' button.

10]Who is it?

<https://www.hackerrank.com/challenges/who-is-it/problem?isFullScreen=true>

HackerRank | Papers | Artificial Intelligence | Natural Language Processing | Who is it?

Problem

Who is it? is an interesting problem in Natural Language Processing, which involves finding out exactly a person corresponds to. This page on the Cornell website has a few basic examples. As a recap, a *person* in linguistics and grammar is a noun or noun phrase like "he"/"she"/"it"/"we" that substitutes for a noun or noun phrase.

Consider the following fragments of text:

Fragment 1

William Jefferson "Bill" Clinton (born William Jefferson Blythe III, August 19, 1946)

Consider the bolded instances of "he" and "his". At all times, refer to William Jefferson "Bill" Clinton.

Fragment 2

Vladimir Putin had a telephone conversation with President of the United States Barack Obama.

The two bolded pronouns are "he" and "his", which correspond to Barack Obama and Putin respectively.

Input Format

The first line will contain an integer N . Including this first line, there are a total of $N+2$ lines in the input file. The first line will be followed by N lines of text, from the snippet of text to be analyzed. This text will contain the names of several characters, places or things. Certain pronouns like "he"/"she"/"it"/"we" will be highlighted by enclosing a pair of "" signs immediately before and after them like ""he"", ""she"", ""it"", ""we"". The second line will contain a list of names or noun phrases (of people, places, objects) from the text, separated by semi-colons. Your task is to identify which of these names, each of the highlighted pronouns correspond to, in the order in which they occur.

Input Constraints

Total number of characters in the given chunks of text will not exceed 20000. Total number of highlighted pronouns in the text segment will not exceed 20.

```
def solve(words):
    for i in range(len(words)):
        if words[i].startswith('"') and words[i].endswith('"'):
            candidates = []
            for noun in nouns:
                length = len(noun.split())
                j = i - length
                while j >= 0 and ' '.join(words[j:j+length]) != noun:
                    j += 1
                if j >= 0:
                    candidates.append(words[j], noun, j)
            results.append(candidates)
    print(*results)
```

Line: 16 Col: 1

Submit Code Run Code Test against custom input

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

Test case 0 Test case 1 Test case 2

Compile Message

Success

Input text: 3

Download