Qualified by andela Assessment Result

Tanyaradzwa Chisepo

Code2040: Required Assessment [2023] Results

✉ tanyachisepo04@gmail.com

Student Remarks

My challenges 1 and 2 are still flagging as incomplete even though I passed all the test cases, I am not sure why

Assessment Summary

100%

11 h, 44 m, 40 s Active Time

- Tanyaradzwa Chisepo **opened** this assessment on Saturday, July 9, 2022 12:31 AM
- Tanyaradzwa Chisepo **started** this assessment on Saturday, July 9, 2022 1:39 AM
- Tanyaradzwa Chisepo **submitted** this assessment after **30 minutes** on Saturday, July 9, 2022 2:09 AM
- This student spent **11 hours and 44 minutes** active in the browser working on the assessment

Solutions Summary

Challenge	Score	Active Time
#1: Inclusive Styleguide: Part 1	100%	3 h, 19 m, 14 s
#2: Inclusive Styleguide: Part 2	100%	2 h, 48 m, 3 s
#3: Inclusive Styleguide: Part 3	100%	5 h, 19 m, 9 s
#4: Inclusive Styleguide: Part 4	100%	18 m, 14 s

Qualified by andela Assessment Code Solution

#1: Inclusive Styleguide: Part 1

Scoring

100%

Timing

3 h, 19 m, 14 s Active Time

5 / 5 Tests (7 Attempts) **775** ms Run Time

Instructions

Background

At Code2040 we work with many companies who are conciously trying to move toward using more inclusive language. In our opinion, there is no more important place to start than the language we use in our codebases and to describe our code. We believe this is a crucial step toward increasing diversity and inclusion in technical spaces. This multipart challenge will be to build a basic delinter built around inclusive language guidelines.

As an example, here (https://developers.google.com/style/inclusive-documentation) is what google's developer documentation style guide has to say about inclusive language, and here (https://mobile.twitter.com/TwitterEng/status/1278733305190342656) is a post from Twitter with some of their internal language guidelines, which we have drawn on as an example for this exercise.

For this task imagine, you have received the following JSON from your team with some style guidelines:

```
GUIDELINES = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
    "suggestions": ["placeholder value", "sample value"]
  },
  {
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]
```

Part 1: Suggestions

Your first task is to write a method that takes a guideline key and returns an array containing the relevant suggestions.

Tips

For the purposes of this task, here's what you need to support:

- The array of suggestions should be in the same order as they are in the json
- It may be useful to include a standard library to parse the JSON, but it is not necessary to do so
- If a the function is called with a key that does not exist in the JSON, return an empty array

Specification

```
{
"method": "suggestions",
"args": {"key": {"type": "String", "desc": "the key for the guideline"}},
"returns": {"type": "Array", "desc": "The suggestions for the guideline"},
"examples": [
  {"args": ["sanity_check"], "returns": ["quick check", "confidence check", "coherence
check"]},
  {"args": ["grandfathered"], "returns": ["legacy status"]},
  {"args": ["you_guys"], "returns": ["you", "you all", "y'all"]},
  {"args": ["not_a_key"], "returns": []}
]
}
```

Solution Code

Python

```
GUIDELINES  = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
```

```
    "suggestions": ["placeholder value", "sample value"]
  },
  {
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]


def suggestions(key):

  for dictionary in GUIDELINES:
    #check to see if given key matches with any of the keys in the dictionary
     if key == dictionary['key']:
        return (dictionary['suggestions'])
  return []
```

Candidate's Tests

```
import unittest
from solution import suggestions
class Test(unittest.TestCase):
  def test_suggestions_should_return_an_array_of_suggestions(self):
    self.assertEqual(suggestions("you_guys"), ["you", "you all", "y'all"]),
    self.assertEqual(suggestions("not_key"), [])

```

#2: Inclusive Styleguide: Part 2

Scoring

100%

3 / 3 Tests (12 Attempts)

Timing

2 h, 48 m, 3 s Active Time

718 ms Run Time

Instructions

Background

At Code2040 we work with many companies who are conciously trying to move toward using more inclusive language. In our opinion, there is no more important place to start than the language we use in our codebases and to describe our code. We believe this is a crucial step toward increasing diversity and inclusion in technical spaces. This multipart challenge will be to build a basic delinter built around inclusive language guidelines.

As an example, here (https://developers.google.com/style/inclusive-documentation) is what google's developer documentation style guide has to say about inclusive language, and here (https://mobile.twitter.com/TwitterEng/status/1278733305190342656) is a post from Twitter with some of their internal language guidelines, which we have drawn on as an example for this exercise.

For this task imagine, you have received the following JSON from your team with some style guidelines:

```
GUIDELINES = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
    "suggestions": ["placeholder value", "sample value"]
  },
  {
```

```
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]
```

Part 2: A Notice

Your second task is to write a function that generates a notice alerting the user that a match has been found and suggesting how they might improve their language.

Tips

Assume that a string has been run through the delinter and a substring has been found that matches one of the non-inclusive language examples in the json. For the purposes of this task, here's what you need to support:

- Follow the pattern for the notice in the specification examples
- The match and each suggestion should each be put within single quotes
- If there is more than one suggestion connect them with the word `or`
- The reference link uses `www.inclusive-styleguide.com` as the base url
- The reference link path is the `key` converted from snake_case to dash-case
- Threre is a single space between the sentences

Specification

```
{
"method": "notice",
"args": {
"key": {"type": "String", "desc": "the key for the guideline"},   "index": {"type": "Integer",
"desc": "the index at which the matching non-inclusive substring begins"}, "match": {"type":
"String", "desc": "the matching non-inclusive substring"}
  },
"returns": {"type": "String", "desc": "The notice generated by the delinter about the non-
inclusive language"},
"examples": [
  {"args": ["sanity_check", 4, "sanity check"], "returns": "<4> Consider replacing 'sanity
check' with 'quick check' or 'confidence check' or 'coherence check'. Reference
https://www.inclusive-styleguide.com/sanity-check for details."},
    {"args": ["man_hours", 8, "woman hours"], "returns": "<8> Consider replacing 'woman hours'
with 'person hours' or 'engineer hours'. Reference https://www.inclusive-styleguide.com/man-
hours for details."},
    {"args": ["grandfathered", 2, "grandfathered"], "returns": "<2> Consider replacing
'grandfathered' with 'legacy status'. Reference https://www.inclusive-
styleguide.com/grandfathered for details."}
]
}
```

Solution Code Python

```python

GUIDELINES  = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
    "suggestions": ["placeholder value", "sample value"]
  },
  {
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]

#function to return the list of suggestions associated with a given key
def suggestions(key):

  for dictionary in GUIDELINES:
     if key == dictionary['key']:
        return (dictionary['suggestions'])
  return []

#function to add single quotes to every item in a string list
def add_quotes(raw_list):

  final = []
  for elem in raw_list:
    result = "'"+ elem + "'"
    final.append(result)
  return final

#function to convert given key from snake_case to dash-case
def dash_case(key):
  result = key.replace("_","-")
```

```
  return result


def notice(key,index,match):

  #loop through the json, search for given key to find specific dictionary
  for dictionary in GUIDELINES:
    if key == dictionary['key']:
      #slice given match string using given index
      string = match[index:]

      #look for sliced match string with search terms in dictionary
      for string in dictionary['search_terms']:

        suggestions = ""
        delimiter = " or "
        suggestions_final = []
        mod_key = ""

        #create list of suggestions in dictionary under 'suggestions' key
        suggestions_list = dictionary['suggestions']
        #add single quotes to every suggestion
        suggestions_final = add_quotes(suggestions_list)
        #add single quotes to the match string
        final_match = "'{}'".format(match)

        #separate the suggestions using "or"
        temp = list(map(str, suggestions_final))
        suggestions = delimiter.join(temp)

        #convert the given key from snake case to dash-case
        mod_key = dash_case(key)


        notice = "<{}> Consider replacing {} with {}. Reference https://www.inclusive-styleguide.com/{} for details.".format(index,final_match,suggestions,mod_key)

  return notice
```

```
101
102
103
104
```

Candidate's Tests

```
1  import unittest
2  from solution import notice
3  class Test(unittest.TestCase):
4    def test_notice_should_return_the_notice(self):
5      self.assertEqual(notice("man_hours",8,"woman hours"), "<8> Consider replacing 'woman
   hours' with 'person hours' or 'engineer hours'. Reference https://www.inclusive-
   styleguide.com/man-hours for details.")
```

#3: Inclusive Styleguide: Part 3

Scoring

100%

5 / 5 Tests (90 Attempts)

Timing

5 h, 19 m, 9 s Active Time

731 ms Run Time

Instructions

Background

At Code2040 we work with many companies who are conciously trying to move toward using more inclusive language. In our opinion, there is no more important place to start than the language we use in our codebases and to describe our code. We believe this is a crucial step toward increasing diversity and inclusion in technical spaces. This multipart challenge will be to build a basic delinter built around inclusive language guidelines.

As an example, here (https://developers.google.com/style/inclusive-documentation) is what google's developer documentation style guide has to say about inclusive language, and here (https://mobile.twitter.com/TwitterEng/status/1278733305190342656) is a post from Twitter with some of their internal language guidelines, which we have drawn on as an example for this exercise.

For this task imagine, you have received the following JSON from your team with some style guidelines:

```
GUIDELINES = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
    "suggestions": ["placeholder value", "sample value"]
  },
  {
```

```
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]
```

Part 3: The Delinter

Your final task is to write the delinting function that provides suggestions about using inclusive language.

Tips

Part 3 is more difficult than the previous two parts. There are edge cases to acccount for and various functions and concepts from the previous two parts to integrate. It's worth noting that completing the first two parts and writing a short note in the upcoming Part 4 is enough to pass this entire assessment. You *can* even skip Part 3 and pass the assessment, but we encourage you to go ahead and try to get the highest score you can get. Also remember partial credit is given on all tasks -- each edge case you solve and test case you pass will increase your score. WE HAVE FULL FAITH IN YOUR ABILITY TO ACE IT!

For the purposes of this task, here's what you need to support:

- Partial matches should not trigger a notice (e.g. `bayou guys` should not trigger the `you_guys` delinter rule)
- There's a 'gotcha' special case to the rule above to watch out for. If you aren't careful `you guys` might match both the `you guys` and the `u guys` search term.
- Matches should be case insensitive
- Use the notice function from Part 2 to output a notice each time you find non-inclusive language
- Each notice will include the index of the first character of the non-inclusive language within the text
- Multiple notices should be displayed in ascending order by index
- If multiple notices are needed connect them with a new line character `\n`
- If no notices should be displayed simply return an empty string `""`
- For an extra challenge, build your function so that if more guidelines were added to the JSON it would continue to work without you having to change your code

Specification

```
{
"method": "delinter",
"args": {
"text": {"type": "String", "desc": "the text to delint"}
  },
"returns": {"type": "String", "desc": "The notice(s) generated by the delinter"},
"examples": [
  {"args": ["Could you guys sanity check my method?"], "returns": "<6> Consider replacing 'you guys' with 'you' or 'you all' or 'y'all'. Reference https://www.inclusive-styleguide.com/you-guys for details.\\n<15> Consider replacing 'sanity check' with 'quick check' or 'confidence check' or 'coherence check'. Reference https://www.inclusive-styleguide.com/sanity-check for details."},
```

```
    {"args": ["I've inserted a dummy value in the block below."], "returns": "<16> Consider replacing 'dummy value' with 'placeholder value' or 'sample value'. Reference https://www.inclusive-styleguide.com/dummy-value for details."},
    {"args": ["This feature is estimated to require 600 engineer hours."], "returns": ""}
]
}
```

Solution Code

Python

```python

import re
GUIDELINES  = [
  {
    "key": "you_guys",
    "search_terms": ["you guys", "u guys", "uu guys"],
    "suggestions": ["you", "you all", "y'all"]
  },
  {
    "key": "man_hours",
    "search_terms": ["man hours", "woman hours"],
    "suggestions": ["person hours", "engineer hours"]
  },
  {
    "key": "grandfathered",
    "search_terms": ["grandfathered"],
    "suggestions": ["legacy status"]
  },
  {
    "key": "dummy_value",
    "search_terms": ["dummy value"],
    "suggestions": ["placeholder value", "sample value"]
  },
  {
    "key": "sanity_check",
    "search_terms": ["sanity check"],
    "suggestions": ["quick check", "confidence check", "coherence check"]
  }
]

#function to return the list of suggestions associated with a given key
def suggestions(key):

  for dictionary in GUIDELINES:
     if key == dictionary['key']:
        return (dictionary['suggestions'])
  return []



#function to add single quotes to every item in a string list
```

```
def add_quotes(raw_list):

  final = []
  for elem in raw_list:
    result = "'"+ elem + "'"
    final.append(result)
  return final

#function to convert given key from snake_case to dash-case
def dash_case(key):
  result = key.replace("_","-")
  return result




def notice_mod(match_list, word_start_index, original_text):

    for word in match_list:
        for number in word_start_index:
          mod_index = number

    #loop through the json, search for given key to find specific dictionary
    for dictionary in GUIDELINES:

      for elem in dictionary['search_terms']:
        if word in dictionary['search_terms']:


          suggestions = ""
          delimiter = " or "
          suggestions_final = []
          mod_key = ""

          #store list of suggestions in dictionary under 'suggestions' key
          suggestions_list = dictionary['suggestions']
          #add single quotes to every suggestion
          suggestions_final = add_quotes(suggestions_list)

          #add single quotes to the match string
          final_match =  "'{}'".format(original_text[mod_index: mod_index + len(word)])

            #separate the suggestions using "or"
          temp = list(map(str, suggestions_final))
          suggestions = delimiter.join(temp)

          #convert the given key from snake case to dash-case
          mod_key = dash_case(dictionary['key'])

          notice = "<{}> Consider replacing {} with {}. Reference https://www.inclusive-
styleguide.com/{} for details.".format(mod_index, final_match, suggestions, mod_key)
```

```
          return notice


def delinter(text):

    original_text = text

    lower_text = text.lower()


    search_list = []
    non_inclusive_list = []
    non_inclusive_index = []
    result = []
    notice =""
    #Loop through all the dictionaries
    for dictionary in GUIDELINES :

      #make a combined list of all the search terms to look for in the string
      for search_thing in dictionary['search_terms']:
        search_list.append(search_thing)

    #check if any search term in search_list exists in the text string
    for elem in search_list:

        #use regex to find exact match
        match = re.search(r'\b' + re.escape(elem) + r'\b', lower_text)

        if match:
            non_inclusive_index.append(lower_text.index(elem))
            non_inclusive_list.append(elem)

            result.append(notice_mod(non_inclusive_list, non_inclusive_index,
original_text))

    delimiter = "\n"
    temp = list(map(str, result))

    #separate the suggestions using "or"
    result = delimiter.join(temp)

    return result
```

```
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
```

Candidate's Tests

```
1  import unittest
2  from solution import delinter
3  class Test(unittest.TestCase):
4    def test_delinter_should_return_the_notice(self):
5      self.assertEqual(delinter("Grandfathered data schema is supported but only because it is grandfathered."), "<0> Consider replacing 'Grandfathered' with 'legacy status'. Reference https://www.inclusive-styleguide.com/grandfathered for details.")
```

#4: Inclusive Styleguide: Part 4

Scoring

100%

1 / 1 Questions

Timing

18 m, 14 s Active Time

Instructions

A Note to a Team Member

Solution Answers

1. Now that you've finished coding, please take no more than ten minutes to write a brief description of your approach to the problem and your thoughts on how the functionality could be further extended. Imagine you are writing an informal note to another team member who works alongside you maintaining this codebase. Use whatever format suits your communication style best. Bullet point lists or loose notes are more than fine. We don't care about perfect spelling or syntax any more than your team member in real life would care--we just want you to talk abour your code in your own voice and help us to better understand it.

   Here are some questions you may want to address:

   - *What design decisions did you make?*
   - *What assumptions did you make?*
   - *What do you think could be improved, refactored, or simplified?*
   - *What other ideas or features would be useful to add?*
   - *Which areas might you ask your team member to help you on?*
   - *Do you think the test coverage could be extended?*
   - *Could you improve clarity with more comments or better variable names?*
   - *Do you think your work on this coding exercise reflects your current technical skills well?*

   -My approach was to break down the problem into smaller tasks based on the tips given

   - I started by simplytrying to print out a notice given a single match term
   - I then moved on to other tasks, such as making it case insensitive, and handling multiple matches, one-by-one
   - I created a "search_list" to store all the search terms from all the dictionaries in the JSON
   - I then iterated through that list and checked each element against the input text string
   - I created a regular expression with word boundaries to avoid a partial match for example with ("you guys") and ("u guys")
   - I created extra functions to add quotes to the suggestions and to convert the match from snake_case to dash-case
   - Resulting codeis too bulky, I think it needs to be optimized. For example, the extra methods may not have been necessay
   - Too many nested loops, it proved to be challenging because I was frequently getting confused by them
   - I also think some of the names may be confusing, but I tried my best to include some commments

   - I think I took too long to complete this challenge because I am not very familiar with the syntax, I have the right idea when it comes to solving the problem, but I lack some of the tools that may make it easier for me to solve it

   0 out of 1 point