CSE143_testout_sample

October 3, 2019

1 CSE 143 Self-assessment, Fall 2019

This testout aims to self-assess your python skill as for you to be successful in this course.

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE`.

- Even if your code work on the given test cases, you can also come up with your own test cases.
- You can write out print statements in your code, to help you test/debug it.

Some of the materials are borrowed from Luca de Alfaro CSE 20

1.1 Preliminaries

Nose. The line below works on colab.research.google.com. If it does not work on your laptop, do:

`pip install nose` if you use pip to install packages, and

`conda install nose` if you use conda

[1]:
```
from nose.tools import assert_equal, assert_true, assert_false

# We check that you are running Python 3.  You need to use Python 3.
import sys
assert_equal(sys.version_info.major, 3)
```

1.2 Question 1

Write a function ziplist that, given two lists, produces a list of pairs, with the first element of the pair coming from the first list, and the second from the second list. The output list of pairs must be as long as the longest of the input lists, and the missing elements from the shorter input list give rise to None in the output list.

[2]:
```
def ziplist(l1, l2):
    if len(l1) < len(l2):
      l1.extend([None] * (len(l2) - len(l1)))
    if len(l1) > len(l2):
```

```
      l2.extend([None] * (len(l1) - len(l2)))
    result = []
    while True:
      try:
        temp = [(l1.pop(0), l2.pop(0))]
        result.extend(temp)
      except IndexError:
        break
    return result
```

[3]:
```
# First, let's check that your code works on lists of equal length.
assert_equal(ziplist(['a', 'b', 'c'], [1, 2, 3]), [('a', 1), ('b', 2), ('c',␣
↪3)])
assert_equal(ziplist([], []), [])
```

[4]:
```
# The function also needs to work on lists of different lengths.
#assert_equal(ziplist([1, 2], []), [(1, None), (2, None)])
assert_equal(ziplist([1], ['a', 'b']), [(1, 'a'), (None, 'b')])
```

1.3 Question 2

Write a function threeormore that, given an input list, outputs the set of list elements that occur 3 or more times.

[5]:
```
def threeormore(l):
    dict_temp = {i:l.count(i) for i in l}
    delete = [key for key in dict_temp if dict_temp[key] < 3]
    for key in delete:
      del dict_temp[key]
    result = set()
    for key in dict_temp:
      result.add(key)
    return result
```

[6]:
```
assert_equal(threeormore([1, 3, 5, 4, 3, 7, 6, 3, 7, 7, 7, 1, 2, 0, 3, 1]), {1,␣
↪3, 7})
assert_equal(threeormore([]), set())
```

1.4 Question 3

[7]:
```
class MyQueue():

    def __init__(self):
        self.queue = []

    def enqueue(self, x):
        self.queue.append(x)
```

```
    def dequeue(self):
        if len(self.queue) < 1:
            return None
        return self.queue.pop(0)
```

[8]:
```
# Example of behavior.
q = MyQueue()
q.enqueue('banana')
q.enqueue('apple')
assert_equal(q.dequeue(), 'banana')
q.enqueue('cherry')
assert_equal(q.dequeue(), 'apple')
assert_equal(q.dequeue(), 'cherry')
assert_equal(q.dequeue(), None)
```

1.5 Question 4

Implement a function olympic_average that, given a list of numbers, if the list is of length at least 3, excludes the minimum and maximum elements and returns the average of the other elements. If the list is shorter than 3, it just returns the average of the elements. If the list is empty, it returns None.

[9]:
```
def olympic_average(els):
    if len(els) > 2:
      els.remove(max(els))
      els.remove(min(els))
    elif els == []:
      return None
    return sum(els)/len(els)
```

[10]:
```
# Let us assume that the list has at least 3 elements.
assert_equal(olympic_average([5, 3, 4, 5]), 4.5)
assert_equal(olympic_average([5, 6, 3, 4, 10]), 5)
```

[11]:
```
# General case.
assert_equal(olympic_average([2, 3]), 2.5)
assert_equal(olympic_average([4]), 4)
assert_equal(olympic_average([]), None)
assert_equal(olympic_average([5, 3, 11, 5]), 5)
```

1.6 Question 5

Implement a function is_anagram that takes as input two strings, and returns True if and only if the strings are anagrams of each other. This means that the both strings are permutations of each other. You can assume that string will only have lowercase letters.

[12]:
```
def is_anagram(w1, w2):
    return sorted(w1)== sorted(w2)
```

[13]:
```
# First, let's assume that the word does not contain repeated letters.
assert_true(is_anagram("home", "hemo"))
assert_true(is_anagram("", ""))
assert_true(is_anagram("figure", "refugi"))
assert_false(is_anagram("seal", "coal"))
```

[14]:
```
# General case, where there can be repeated characters.
assert_true(is_anagram("banana", "ananab"))
assert_false(is_anagram("hello", "helo"))
assert_false(is_anagram("banana", "bnana"))
```

1.7 Question 6

Read from file sample_text.txt and save each line as a list of words. Return a list of list as the content of the input file.

[15]:
```
def read_file(filepath):
    result = []
    with open(filepath, "r") as f:
        for line in f:
            result.extend([line.split()])
    return result
```

[16]:
```
# Keep the sample_text.txt in the same directory as this notebook
sample_text = read_file('sample_text.txt')
assert_equal(sample_text, [['a', 'crow', 'was', 'sitting', 'on', 'a'],
                           ['branch', 'of', 'a', 'tree', 'with', 'a'],
                           ['piece', 'of', 'cheese', 'in', 'her', 'beak'],
                           ['when', 'a', 'fox', 'observed', 'her', 'and'],
                           ['set', 'his', 'wits', 'to', 'work', 'to'],
                           ['discover', 'way', 'of', 'getting', 'the',␣
 ↪'cheese']])
```

1.8 Question 7

You don't need to know anything about word vector but you need to be able to read documentation and figure out how to use other library/module. You need to use pip or conda to install these libraries before you could use them. 1. Use gensim library to train a word vector model using sample_text you created above. Read more about the fucntion here https://radimrehurek.com/gensim/models/word2vec.html. 2. Implement a function to get the vector of a given word. Pass size=10, min_count=1 to as parameters to train the model. Return None if the word is not in the vocabulary. 3. Use Sklearn library to implement a function to calculate the cosine similarity (round to two precision) and euclidean distance between two vectors (round to two precision). Return none if any word is not in the vocabulary. Read here

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

[17]:
```
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances

def get_word_vector(word):
    model = Word2Vec(sample_text, size=10, min_count=1)
    return model.wv[word]

def calculate_cosine_similarity(word1, word2):
    if word1 == word2:
        return 1.00
    vec1 = get_word_vector(word1)
    vec2 = get_word_vector(word2)
    return cosine_similarity([vec1], [vec2])

def calculate_euclidean_distance(word1, word2):
    if word1 == word2:
        return 0.00
    vec1 = get_word_vector(word1)
    vec2 = get_word_vector(word2)
    return euclidean_distances([vec1], [vec2])

assert_equal(len(get_word_vector('fox')), 10)
assert_equal(calculate_cosine_similarity('tree', 'tree'), 1.00)
assert_true(calculate_cosine_similarity('branch', 'tree'), -0.46)
assert_true(calculate_euclidean_distance('branch', 'tree'), 0.14)
```