CSE 331 **Spring 2018**

Project 7

100 points
Due April 12th by 8:00pm
The Author of this project is: ***Conner Bean***
This is not a team work, do not copy somebody else's work.

Assignment Overview

You will be creating a hash table class that resolves collisions with separate chaining, as well as using the newly created hash table class to find all words in a string that appear a specific amount of times. You are given a skeleton of a hash table and linked list with methods to finish. You are also given a complete HashListNode class, do not alter this class.

Assignment Deliverables

Be sure to use the specified file name(s) and to submit your files for grading **via D2L Dropbox** before the project deadline.

- HashTable.py
- LinkedList.py

Assignment Specifications

Your task will be to complete the methods listed below.

For HashTable:

- **__repr__:**
  - Returns string representation of Hash Table. This is solely for your personal debugging use. You can follow example output format or create your own.
- **insert(key, value) :**
  - Inserts key and value into hash table, location based off of key hash from the provided hash_function(key). The key is *guaranteed* to be a string. If key is already in table, reassigns value of old node to new value. Does not allow empty strings/None values to be inserted into the hash table. Does not return anything. If load factor is greater than 0.75 *after* insertion, doubles table size.
- **find(key) :**
  - Returns *HashListNode* in table with specific key. If key is not found in table, returns False.
- **lookup(key) :**
  - Returns the *value* of the Node in the table with specific key. If key is not found in table, returns False.

- **delete(key) :**
    - Deletes the HashListNode with specific key from hash table. Does not return anything.
- **double() :**
    - Doubles the size of hash table, rehashes values. Does not return anything.
- **rehash() :**
    - Rehashes all keys of hash table. Does not return anything.

For LinkedList:

- **__repr__ :**
    - Returns string representation of linked list. This is solely for your personal debugging use. You can follow example output format or create your own.
- **append(key, value) :**
    - Pushes new node with key and value to the end of linked list. Does not return anything.
- **remove(key) :**
    - Removes node with specific key from linked list. Does not return anything.
- **find(key):**
    - Returns node with key if present in linked list. Otherwise returns false.

Finally:

- **def FindWords(phrase, k) :**
    - Parses through a string (phrase), utilizing HashTable class in order to find substrings of the phrase that appear *exactly* k times, case insensitive (i.e. "one" == "OnE"). Returns a ***lowercase*** **set** of all substrings that appear k times. All substrings in set should be cast as lowercase before being appended, order does not matter. Each substring should be of length greater than 1. See sample output for more details. NO string methods may be used in this function, aside from the operator method str[x].

You can make additional helper functions, if useful.

Assignment Notes

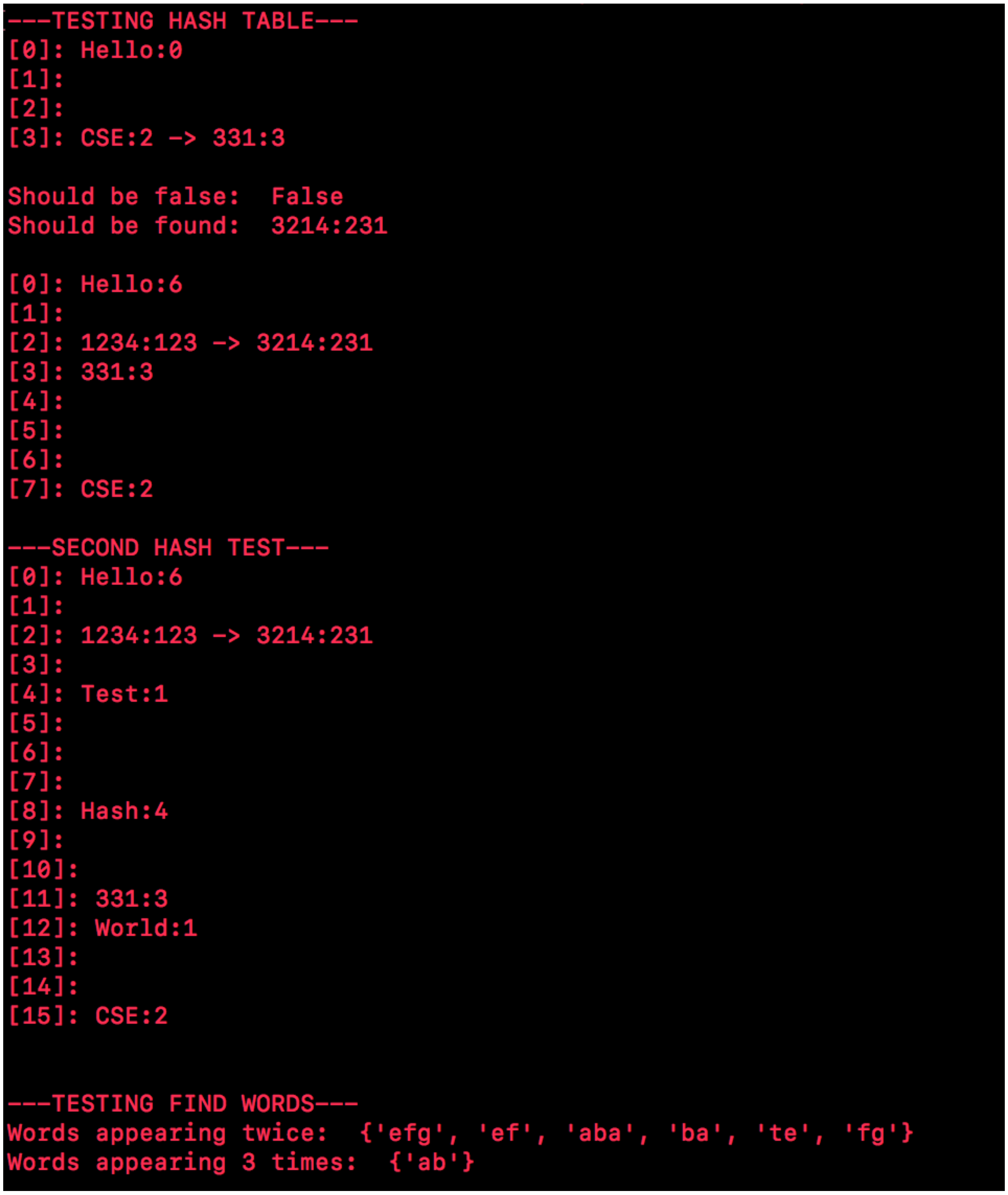
The use of auxiliary containers are strictly prohibited. This includes python dictionaries, sets, tuples, and lists. Use of either dictionaries or lists will result in 75% reduction. The use of string methods (outside of the operator str[x]) in the FindWords function is also prohibited and will result in similar grade reduction.

Points will be deducted if your solution has any warnings of type:

- The newest distribution python 3.6 interpreter will be used to execute your solution.
- FindWords function should run in O(n^2) time.
- Insertion into hash table should run in O(1) time.
- Find/Lookup should run in O(k) time, where k is average length of linked list in hash table.
- Delete should run in O(k) time, where k is the average length of linked list in hash table.
- The hash list node should not be edited in anyway.
- You are required to complete the docstrings for any unmade and created function signatures.
- To test your classes, main.py is provided. Compare your results to the output below. It is recommended you also create test cases yourself in order to test for various edge cases.
- Errors when using your solution that cause the grading script to fail will result in a 25% deduction.
- You may not change any function signatures in anyway, which include class definitions.
- Your solution will be ran against 7 testcases checking for various edge cases against your solution.
- Your solution will be graded and tested against the equivalent equality operators and **not** standard output.

Testing your work

Run your project on Pycharm, see sample run below of **main.py**



```
---TESTING HASH TABLE---
[0]: Hello:0
[1]:
[2]:
[3]: CSE:2 -> 331:3

Should be false:  False
Should be found:  3214:231

[0]: Hello:6
[1]:
[2]: 1234:123 -> 3214:231
[3]: 331:3
[4]:
[5]:
[6]:
[7]: CSE:2

---SECOND HASH TEST---
[0]: Hello:6
[1]:
[2]: 1234:123 -> 3214:231
[3]:
[4]: Test:1
[5]:
[6]:
[7]:
[8]: Hash:4
[9]:
[10]:
[11]: 331:3
[12]: World:1
[13]:
[14]:
[15]: CSE:2


---TESTING FIND WORDS---
Words appearing twice:  {'efg', 'ef', 'aba', 'ba', 'te', 'fg'}
Words appearing 3 times:  {'ab'}
```