Test-Driven Development

Due Oct 10 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** zip
Available Oct 7 at 12am - Oct 10 at 11:59pm 4 days

This assignment was locked Oct 10 at 11:59pm.

Overall goal:

Using Test Driven Development (TDD), implement in Python a function kwic (Key Word in Context, you can look up this problem online, and read David Parnas' famous essay if you want).

Specification of kwic:

The function kwic, given a string of arbitrary length:

- breaks the string into lines (by line break "\n")
- breaks the lines into words by Python whitespace (string.split() or .isspace() definition of whitespace)
- produces the circular shifts for each line, e.g. from "the point of software engineering is to build stuff" would produce:
"point of software engineering is to build stuff the"
"of software engineering is to build stuff the point"
"software engineering is to build stuff the point of"
...
- associate each set of shifts (including the "0-shift") with the index of the starting line in the initial breakdown of lines (in the form of a tuple)

- alphabetizes the total set of shifted lines (ignore case but not punctuation marks).
- returns this alphabetized "index"

- if the optional argument "ignoreWords" is present, does not include shifts whose first word is in the set (ignore case and the punctuation marks [".", "?", ",", "!", ":"]

- if the optional argument "listPairs" is set to True, also return a list of all pairs of words that appear in a line together more than once. kwic returns a tuple, in this case, with the "index" list first, then the list of all word pairs. The words in the word pairs should be sorted alphabetically (both in each pair, and over all pairs), converted to lowercase, and stripped of the punctuation marks in the set above), and the number of lines pairing the words should be the last element. A word must have at least one character in it. A word pair list might be:

```
[(("a","dragon"), 5),
 (("a","rowboat"), 2),
 (("engineering","software"), 1245)]
```

though that would be a pretty strange document.

- finally, if the optional argument "periodsToBreaks" is set to True, then line breaking should break a "line" anywhere there is an lowercase alphabet character (a..z) followed by a period, followed by a space, instead of using "\n"

For example, kwic("Design is hard.\nLet's just implement.") should produce:

```
[(["Design","is","hard."], 0),
 (["hard.","Design","is"], 0),
 (["implement.","Let's","just"], 1),
 (["is","hard.","Design"], 0),
 (["just","implement.","Let's"], 1),
 ...
```

but kwic("Design is hard.\nLet's just implement.", ignoreWords=["is"]) would produce:

```
[(["Design","is","hard."], 0),
```

```
 (["hard.","Design","is"], 0),
 (["implement.","Let's","just"], 1),
 (["just","implement.","Let's"], 1),
 ...
```

and kwic("Design is hard.\nLet's just implement.", ignoreWords=["is"], listPairs=True) would produce:

```
([(["Design","is","hard."], 0),
 (["hard.","Design","is"], 0),
 (["implement.","Let's","just"], 1),
 (["just","implement.","Let's"], 1),
 ...
 [])
```

Finally, for a larger example,

```
kwic.kwic("Hello there.\nHello there, buddy.\nHello and goodbye, buddy.\nHello is like buddy Goodbye!", listPairs=True) ==
 ([(['and', 'goodbye,', 'buddy.', 'Hello'], 2), (['buddy', 'Goodbye!', 'Hello', 'is', 'like'], 3), (['buddy.', 'Hello', 'and',
 'goodbye,'], 2), (['buddy.', 'Hello', 'there,'], 1), (['Goodbye!', 'Hello', 'is', 'like', 'buddy'], 3), (['goodbye,', 'budd
y.', 'Hello', 'and'], 2), (['Hello', 'and', 'goodbye,', 'buddy.'], 2), (['Hello', 'is', 'like', 'buddy', 'Goodbye!'], 3), (['H
ello', 'there,', 'buddy.'], 1), (['Hello', 'there.'], 0), (['is', 'like', 'buddy', 'Goodbye!', 'Hello'], 3), (['like', 'budd
y', 'Goodbye!', 'Hello', 'is'], 3), (['there,', 'buddy.', 'Hello'], 1), (['there.', 'Hello'], 0)], [(('buddy', 'goodbye'), 2),
 (('buddy', 'hello'), 3), (('goodbye', 'hello'), 2), (('hello', 'there'), 2)])
```

and kwic of

```
kwic.kwic ("Hello there.  Hello there, buddy.    Hello and goodbye, buddy. Hello is like buddy Goodbye!", listPairs=True, peri
odsToBreaks=True)
```

should have the same result.

Development Instructions (TDD Aspect)

You should develop kwic in a series of versions, kwic0.py through kwicN.py (where N is the number of versions, at least 9) such that for each kwic<n>.py there is a corresponding file testkwic<n>.py that is a test for the functionality added in that version. Do not import any non-standard library Python modules. Write Python 2.7.11 legal code.

kwic<n> should never pass testkwic<n+1> (or greater), but should pass all tests up through testkwic<n>. Use Python assert to implement the test. No need to worry about unit test frameworks for now, you can just write each test as straight Python that assigns variables, imports kwic, and checks results. The testkwics should import kwic, not kwic<n>. E.g., a simple test would be:

```
import kwic
```

```
document = ""
assert(kwic.kwic(document) == [])
```

(I'm giving you this one for free, if you want it)

There should be no lines of code in any kwic<n>.py that are not covered by some test.

In each testkwic<n> comment on what the test is testing, and how this relates to the requirements above.

In each kwic<n> comment on what you are adding, how you refactored the code, and the state of the overall design.

Your code should also pass my secret testing regime.

kwicN.py should also have a copy named kwic.py, and should pass all tests. I'll use kwic.py in my testing, and may also check that earlier versions of your code fail my tests, in general.

Your writeup (in kwic.txt) should answer the following questions:

1. How did you decide what to test first? Would your final code change significantly if you changed the order of tests?

2. What did you think of test driven development, for this problem? What are the strengths and weaknesses of the approach? Does it encourage/discourage certain kinds of program designs?

Writeup should be at least 500 words.

Up to 3 points of the 20 point grade will be assigned based on code coverage of your tests and the number of other student implementations that fail one of your tests (only if the test is actually correct and my reference version passes the test, of course).

Submit your files via canvas as <youronidlogin>361assign1.zip, where the zip file contains the Python and .txt files described above.

Final Hints

```
>>> print kwic.kwic(". . a")
```

```
[(['.', '.', 'a'], 0), (['.', 'a', '.'], 0), (['a', '.', '.'], 0)]
```

```
>>> print kwic.kwic(". . a", periodsToBreaks=True)
```

```
[(['.', '.', 'a'], 0), (['.', 'a', '.'], 0), (['a', '.', '.'], 0)]
```

```
>>> print kwic.kwic(". A B\n. A B C\n. A B C D", listPairs=True)
```

```
([(['.', 'A', 'B'], 0), (['.', 'A', 'B', 'C'], 1), (['.', 'A', 'B', 'C', 'D'], 2), (['A', 'B', '.'], 0), (['A', 'B', 'C',
 '.'], 1), (['A', 'B', 'C', 'D', '.'], 2), (['B', '.', 'A'], 0), (['B', 'C', '.', 'A'], 1), (['B', 'C', 'D', '.', 'A'], 2),
 (['C', '.', 'A', 'B'], 1), (['C', 'D', '.', 'A', 'B'], 2), (['D', '.', 'A', 'B', 'C'], 2)], [(('a', 'b'), 3), (('a', 'c'),
 2), (('b', 'c'), 2)])
```