Test Driven Development of BibTEX Author Strings

Introduction

BibTEX[1] is a reference management system that is used with LaTEX to ease the use of citations in documents. A BibTEX database is a text file containing bibliographic information. A typical entry might look like this:

```
@Article{hell:duality_tree_homo,
  author =   {Hell, P. and Ne\v{s}et\v{r}il, J. and Zhu, X. },
  title =   {Duality and Polynomial Testing of Tree Homomorphisms},
  journal =   {Transactions of the American Mathematical Society},
  year =   {1996},
  volume =  {348},
  number =  {4},
  pages =  {1281-1297}
}
```

For the purpose of sorting the entries in the bibliography it is important to know the surname and forenames of an author. BibTEX provides a convention where surname and forenames can be unambiguously given. In general there are two ways of expressing a name.

```
   Surname, Forename1 Forename2
```

or

```
   Forename1 Forename2 Surname
```

The reason for this complication is that `Ludwig van Beethoven` has the surname `van Beethoven`. So this can be expressed in BibTEX either as:

```
van Beethoven, Ludwig
```

or as:

```
Ludwig {van Beethoven}
```

A list of names is separated by `and`.

[1] `http://en.wikipedia.org/wiki/BibTeX`

Goal

The goal of this lab is to write a python function `extract_authors` that takes a string of names separated by `and` and return a list of pairs or strings `('Surname','Forenames')`. You will do this by **test driven development (TDD)**. Initially, you will develop a function `extract_author` that takes a single author and returns a single pair `('Surname','Forenames')`. You can use this function to implement `extract_authors`.

Lab Instructions

You will develop a Python module named `bibtex` containing the functions `extract_author(str)` and `extract_authors(str)` such that all the tests given below pass. You must develop code to pass the tests one by one, *in the order given*.

As you develop your code you need to keep a code diary. In a separate text file paste each test and the code that you write to pass the test. It is important that you follow **TDD** *strictly*. Only write enough code to pass the next test, and refactor when necessary.

There is nothing to hand in with this lab. You will be orally marked by one of the lab assistants. You will show your code diary that you have produced, and the lab assistants will ask you questions about the code. The lab can be done in pairs, but both members of the group are required to understand the code that you have written.

Test Cases

The following listing contains all the test cases that your code needs to pass. Note that you are expected to type in the test cases yourself.

```
import bibtex
import unittest
class TestAuthorExtract(unittest.TestCase):
  """ Functions for testing the extraction of author names.

  Note that the rules for extracting bibtex author names is actually
  quite complicated.  As these test cases illustrate. I haven't
  implemented all the cases. """

  def setUp(self):
      self.simple_author_1 =  "Smith"
      self.simple_author_2 =  "Jones"
      self.author_1 = "John Smith"
      self.author_2 = "Bob Jones"
      self.author_3 = "Justin Kenneth Pearson"
```

```
        self.surname_first_1 = "Pearson, Justin Kenneth"
        self.surname_first_2 = "Van Hentenryck, Pascal"
        self.multiple_authors_1 = "Pearson, Justin and Jones, Bob"
    def test_author_1(self):
        #Test only surnames.
        (Surname,FirstNames) = bibtex.extract_author(self.simple_author_1)
        self.assertEqual( (Surname,FirstNames) , ('Smith','') )
        (Surname,FirstNames) = bibtex.extract_author(self.simple_author_2)
        self.assertEqual( (Surname,FirstNames) , ('Jones','') )
    def test_author_2(self):
        #Test simple firstname author.
        (Surname,First) = bibtex.extract_author(self.author_1)
        self.assertEqual( (Surname,First) , ("Smith","John"))
        (Surname,First) = bibtex.extract_author(self.author_2)
        self.assertEqual( (Surname,First) , ("Jones","Bob"))
    def test_author_3(self):
        (Surname,First) = bibtex.extract_author(self.author_3)
        self.assertEqual( (Surname,First) , ("Pearson","Justin Kenneth"))
    def test_surname_first(self):
        (Surname,First) = bibtex.extract_author(self.surname_first_1)
        self.assertEqual( (Surname,First) , ("Pearson","Justin Kenneth"))
        (Surname,First) = bibtex.extract_author(self.surname_first_2)
        self.assertEqual( (Surname,First) , ("Van Hentenryck","Pascal"))
    def test_multiple_authors(self):
        Authors = bibtex.extract_authors(self.multiple_authors_1)
        self.assertEqual (Authors[0] , ('Pearson','Justin'))
        self.assertEqual (Authors[1] , ('Jones', 'Bob') )
if __name__ == '__main__':
  unittest.main()
```

Hints

There are a lot of helpful functions in Python for string handling. You should look at `join`,`split` and `strip`. Also some of the string handling exercises included in the pre-lab might be useful.