Lab 2

Instructions:

1. **Read the HW guidelines** posted in d2l.depaul.edu > admin.

2. Your solution file MUST be named **lab2.py**. In a comment, include the name(s) of any collaborators.

3. To receive full credit, the **names** of files, functions and the **output** must be **exactly** as indicated here.

4. **Test your code** by downloading the file **lab2TEST.py** in the same working folder and typing in the shell:

```
if __name__=='__main__':
    import doctest
    print( doctest.testfile('lab2TEST.py') )
```

Problems:

1. Write an `Animal` class (this is based on the example from Chapter 8, and you are welcome to start with what is included there, you will need to make modifications). Your object is to make this client code work:

```
>>> a = Animal()
>>> a
Animal('default','default',0)
>>> a.speak()
I am a 0 year-old default and I default.
>>> a.setSpecies('tiger')
>>> a.setLanguage('growl')
>>> a.setAge(8)
>>> a
Animal('tiger','growl',8)
>>> a.speak()
I am a 8 year-old tiger and I growl.
>>> b = Animal('jackalope','can imitate anything',33)
>>> b
Animal('jackalope','can imitate anything',33)
>>> b.speak()
I am a 33 year-old jackalope and I can imitate anything.
>>> [b.speak()]   # make sure print not return
I am a 33 year-old jackalope and I can imitate anything.
[None]
```

```
TEST for __repr__
note that quotes should appear around
the species and language but not the age

>>> eval(str(a))
Animal('tiger','growl',8)
>>> eval(str(b))
Animal('jackalope','can imitate anything',33)
```

An `Animal` object has three variables:

- `species - str`
- `language -str`
- `age - int`

The `Animal` class has the following methods:

a. `__init__` - can create Animals either by specifying no arguments (see above) or by specifying the species, language and age.
b. `__repr__` – returns str representation of Animal. See client code. Note for full credit, eval statements should work.
c. `setSpecies` – sets species
d. `setLanguage` – sets language
e. `setAge` - takes an integer as a parameter and sets the animal's age to the value of the parameter
f. `speak` – prints message (see sample client code)

2. In the same module/file, write a function `processAnimals` that takes one argument, the name of an input file. The function **returns the list** of animals created or an empty list if the file didn't contain any lines. The input file contains zero or more lines with the format: species, language, age where species is a string representing an animal's species, language is a string representing an animal's language, and age is an integer representing an animal's age. The items on each line are separated by commas.
The `processAnimals` function should read all lines in the file, creating an Animal object for each line and placing the object into a list. The function **also prints** to output the result of calling the method `speak` on each object in the list it created. Don't forget to close the input file. The information below shows how you would call the function `processAnimals` on an example file. Please note that your implementation should work on any file with the specified format, not just the one provided as a test case:

```
>>> processAnimals('animals.txt')
I am a 8 year-old cat and I meow.
I am a 22 year-old dog and I bark.
```

```
I am a 2 year-old bat and I use sonar.
[Animal('cat','meow',8), Animal('dog','bark',22),
Animal('bat','use sonar',2)]
>>> processAnimals('animals.txt')
I am a 8 year-old cat and I meow.
I am a 22 year-old dog and I bark.
I am a 2 year-old bat and I use sonar.
[Animal('cat','meow',8), Animal('dog','bark',22),
Animal('bat','use sonar',2)]
>>>
```