CSCI 232: Data Structures: Project 4

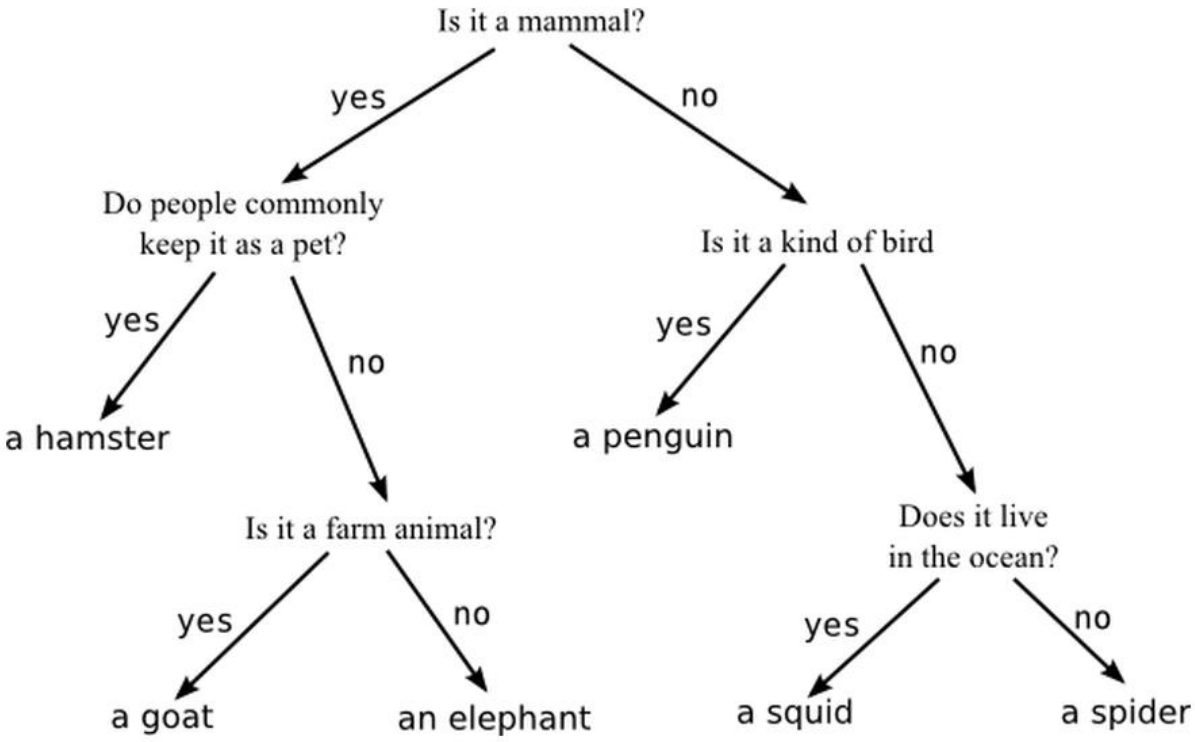
Due Sunday, December 7 at 11:55 pm

Problem Description – Animal Guessing Game

This project will require the implementation of an animal guessing game using a **Binary Tree**. You will setup a binary tree that will hold a series of yes/no questions about animals. The leaves of the tree will store the names of the animals.

The game will work very much like 20 questions. The user thinks of an animal. The application then asks questions to try to guess the animal you are thinking of. If the application guesses incorrectly, you provide more knowledge to the program so that next time the game is played, the program is a bit smarter.

After several rounds of the game, the tree could look like this:



When it is first run, the application doesn't know any animals. It will simply give up and ask you to tell it the answer (user input shown is **bold**):

```
Please think of an animal.
I give up.
What animal is it?
Dog
What question should I have asked?
Does it have fur?
```

After several games have been played, the following dialog might occur:

```
Please think of an animal.
Q: Does it have fur?
N
Q: Does it have tusks?
Y
Q: Does it have big ears?
Y
Is it an elephant?
N
Oops, I guessed wrong.
```

Notice that the computer's strategy is to ask all the relevant questions possible before guessing an animal. (It does NOT guess an animal then ask another question ...) If the computer guesses incorrectly, or runs out of things to guess, the user is asked to provide the correct animal and a question which would identify that animal:

```
I give up.
What animal is it?
Wild Boar
What question should I have asked?
Does it sing Hakuna Matata?
```

You should limit the user's response to questions to a single character 'y' or 'n' (upper or lower case) or 'q' to quit.

See http://www.animalgame.com/ for an example of how this game plays.

Implementation

Use the provided `BinaryTree.py` module as the starting point of your implementation (you may have to modify the binary tree to suit your needs). Then, create a module called `Animal.py` that will contain your game logic.

The first thing your application should do is read in the current tree data from a file named `data.txt` to populate the binary tree. If the file is not found, start with an empty tree. This file will be updated each time the user exits the program, making the application smarter (if the file wasn't found, create it upon exiting). I'm leaving it up to you to figure out how you want to store your binary tree in a file and read it into your application.
(*As a hint, consider writing a preorder traversal of the tree with some kind of indication of which nodes are leaf nodes to your file. This traversal can then be read and a tree can be created.*)

Each round of the game in which an incorrect guess is made by the computer will result in two new nodes (the new animal and the animal guessed) being added to the tree and a change in the contents and the children of the node used as a guess in that round by the computer. The new question will be added to a node and the original and new animals will be added as the leaves of that node. Each round of the game in which a correct decision is made will result in no new nodes being added to the tree. Each round restarts the guessing game at the root.

When the user exits the application (by pressing q), save the binary tree to the `data.txt` file.

I recommend you start by implementing everything except the logic to read/write the binary tree to the file. The application will just always start with an empty tree and begin from there. Once you have that working, add the ability to write to the file upon exiting and read from the file when starting.

Final Deliverable

The **deliverable** will be a compressed zip file containing the following files:

- `binaryTree.py` **# Implementation of the binary tree**
- `animal.py` **# Contains the game logic**
- `dataLogic.py` **# Contains the information to read a file to populate a tree and write a tree to a file**
- `data.txt` **# Contains your prepopulated game data of at least 10 questions and the corresponding animals**