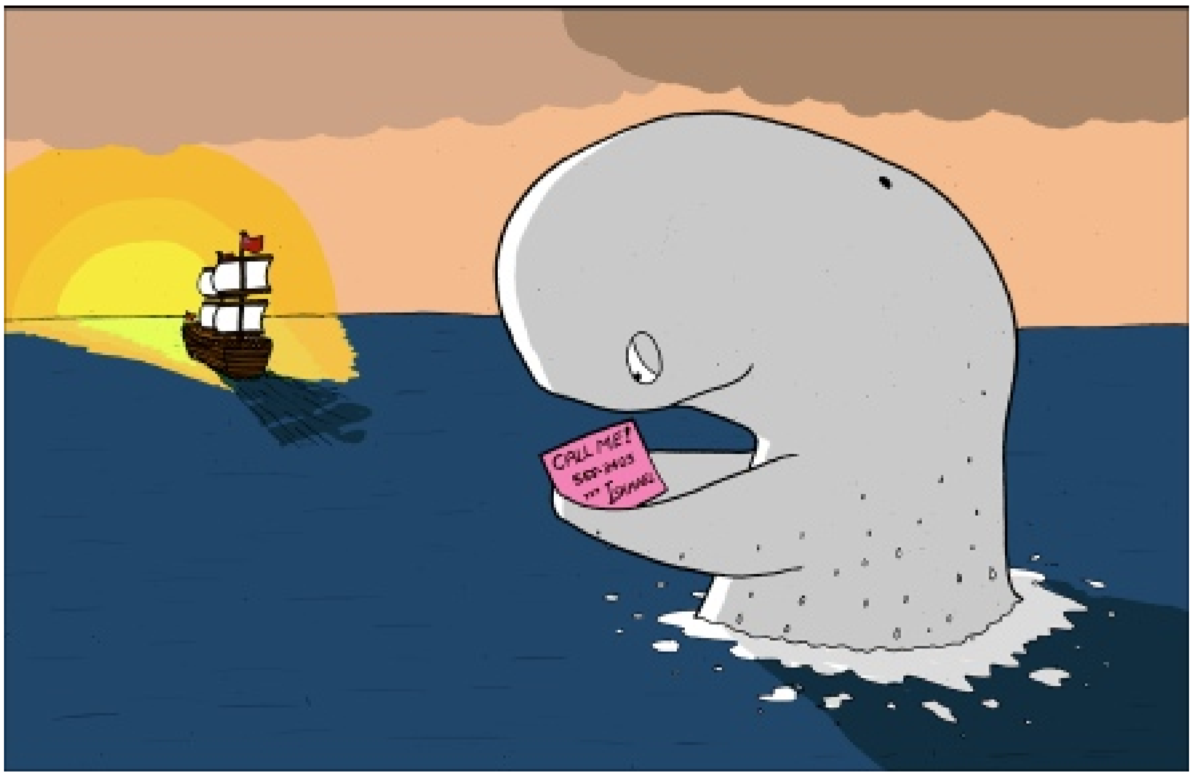
CS 312 - Assignment 1
Ruby Visits the Zoo

Due 11:59pm on Monday, February 20



http://www.lukesurl.com/archives/967

In this assignment we will play a guessing game (somewhat similar to 20 questions), with the computer doing the guessing—and learning at the same time. In the sample below, the human's responses are shown in red.

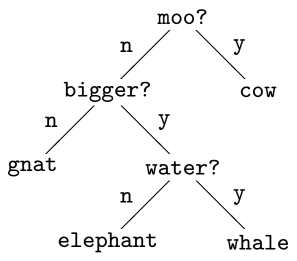
```
Welcome to the Animals game!

Shall we play a game? y
Were you thinking of an elephant? n
Doh! What was the animal? cow
What question separates cow from elephant? Does it moo?
What is the correct answer for cow? y

Shall we play a game? y
Does it moo? y
Were you thinking of a cow? y
Great!

Shall we play a game? y
Does it moo? n
Were you thinking of an elephant? n
Doh! What was the animal? gnat
What question separates gnat from elephant? Is it bigger than a breadbox?
What is the correct answer for gnat? n

Shall we play a game? y
Does it moo? n
Is it bigger than a breadbox? y
Were you thinking of an elephant? n
Doh! What was the animal? whale
What question separates whale from elephant? Does it live in water?
What is the correct answer for whale? y

Shall we play a game? n
Bye!
------------------------------
Your game tree was:
Question: Does it moo?
Question: Is it bigger than a breadbox?
```

```
Answer: gnat
Question: Does it live in water?
Answer: elephant
Answer: whale
Answer: cow
```

Strategy

The program maintains a binary tree whose internal nodes contain questions and whose leaves contain the names of animals. The left and right children of an internal node correspond to the responses "no" and "yes" (left being no and right be yes). When the program makes a wrong guess, it collects enough information to update the tree.

Notice that the tree has the property that a node is either a leaf or has two children. That property makes it easy to distinguish answers from questions. It also allows us to print out the game tree unambiguously with the nodes listed in *preorder*. Here is a picture and the corresponding textual representation of a simple tree.



```
Question: Does it moo?
Question: Is it bigger than a breadbox?
Answer: gnat
Question: Does it live in water?
Answer: elephant
Answer: whale
Answer: cow
```

The lines printed out follow an preorder traversal of the tree. A question is prefixed `Question: ` and an answer by `Answer: `. These should not be explicitly stored in the tree, but can be derived based on whether the node is a leaf or not.

Requirements

- You must write a `node` class for this assignment to represent the binary tree. This class must be in a separate file from the rest of your implementation for handling the game interaction.

  - Your class must have a `to_s` method that prints out the preorder traversal of the tree.
  - Your class must implement one additional method that you use besides `to_s` and the constructor/initializer.

- Any capitalization variant of 'y' or 'yes' should be considered as an affirmative response and everything else negative.
- Your code should change between 'a' and 'an' appropriately when asking "Where you thinking of ...", though you do not need to handle words starting with 'y'.
- Your program should normalize the question input by the user so that it: does not have any surrounding whitespace, ends in a question mark, and starts with a capital letter. For example if the user enters "does it moo ", you would normalize this to "Does it moo?".

Extra Credit

You may earn up to 2 points of extra credit by adding one of the additional features listed below. If you add these features, include a note in the comments of your main file (`animal_game.rb`).

- **[1 point]** Allow the user to specify a filename containing an animal game tree (i.e. copied and pasted from the tree printout). Read this in and start the game from this tree, rather than starting with just asking if the animal is an elephant.
- **0.5 points**] Require the user to enter capitalized variants of "n" or "no" for no and prompt the user to enter the answer again if the user doesn't enter a yes or no answer.
- [**? points**] Pick some other way of improving the game. Do NOT alter the general input/output of the game. If you have a question of the appropriateness of an add-on, please come see me.

Style/Comments

Make sure that your program is properly commented and that you've used good style.

Comments

- All classes and methods should have a block of comments above them.
- Each file should have a block of code at the top with a brief description, your name and the date.
- You should comment any tricky or confusing things in the code.

Style

- You should use appropriate `attr_` tags when defining class instance variables.
- You should appropriately use ? and ! in naming functions.
- You code should be succinct and follow the general ruby conventions, including variable, method and class naming schemes.
- Your code should be broken down into appropriate functions and blocks of code.

When you're done...

Make sure you've followed the specifications above and check that your input/output match the example above, including checking that your output tree is in the same order.

To submit, follow the instructions on the course web page. Your main program should be in a file called `animal_game.rb`.

Grading

You will be graded based on the following criteria:

criterion		points
Functionality		
	proper game play	20
	prints game tree properly	5
	handles affirmative properly	4
	question normalization	3
	a vs. an	1
Meets specifications		2
Style and commenting		5
Extra credit		2
Total		40 (+2)