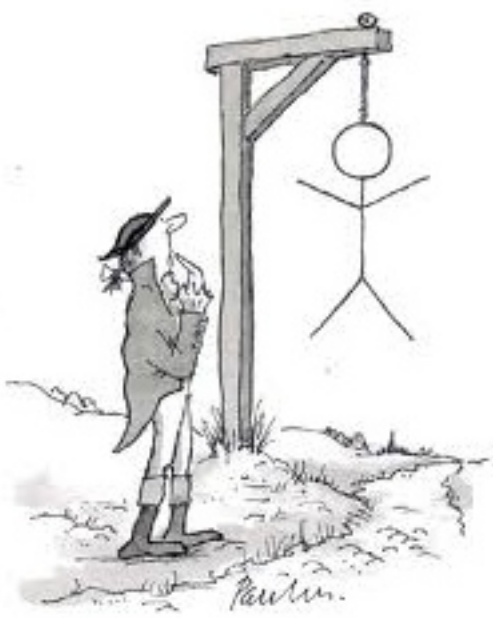
CS51A - Assignment 8
Movie Hangman

Due: Sunday, April 6th at 11:59pm



For this assignment, you will be *re-implementing* hangman using an object-oriented approach (i.e., using a `class` to keep track of the state of the game). You are allowed (and should) reuse code where appropriate from your original hangman version. The point of this is not to rewrite that code, but to view the problem from a different perspective.

Starter

Like many of the previous assignments, this assignment also has a starter code that contains two things:

- The movie database from Assignment 4. It's the same one, but we wanted to save you the hassle of copying it over (or re-downloading it).

- A starter file called `assign8.py` that has some initial code to get you started.

Create a directory called `assignment8` and put the contents of the starter file in that directory (see Assignemnt 4 for more detailed instructions on how to extract a starter if you can't remember).

`http://www.cs.pomona.edu/classes/cs51a/assignments/assign8-starter.zip`

Put your name(s), etc. at the top of the file in comments.

Warmup

Before we dive into hangman, we're first going to practice writing a simpler class. In your .py file above the hangman code (but below the imports) create a class called `LabeledExample` that represents a labeled text example (like from the last assignment). The class should have the following methods:

- A constructor that takes two arguments in this order: a string named `line` representing a line of text and a boolean indicating whether it is a positive example. If the boolean is `True` then the example is positive.
- `is_positive`, which takes zero arguments and returns a boolean indicating whether the example is positive or not.
- `lowercase`, which takes zero arguments and *changes* the example to be lowercased. It does not return anything.
- `get_words`, which takes zero arguments and returns a `list` of the words in the example (words are determined by splitting the text up based on whitespace and don't necessarily have to be unique).
- `contains_word`. The method should take a word as input and return a boolean indicating whether or not that word exists in the text of the example.

  Hint: the `in` operator can be used to see if a value occurs in a list of things. For example:

  ```
  >>> 1 in [1, 2, 3, 4]
  True
  ```

- `__str__`. Returns the string representation of the example, which should be the text followed by a tab, followed by either "positive" or "negative".

Here is an example of the class being used:

```
>>> l1 = LabeledExample("This is great", True)
>>> l2 = LabeledExample("This is bad", False)
>>> l1.is_positive()
True
>>> l2.get_words()
['This', 'is', 'bad']
>>> l2.lowercase()
>>> l2.get_words()
['this', 'is', 'bad']
>>> print(l1)
This is great positive
>>> print(l2)
this is bad negative
>>> l2.contains_word("bad")
True
>>> l1.contains_word("bad")
False
```

When designing any class, think about what data you need to store and how you're going to store it (specifically, what *instance variables* your going to use).

Hangman revisited

For the rest of the assignment, we're going to be reimplementing hangman to use an object-oriented approach. Specifically, we're going to keep track of all of the game information in a class called `Hangman`. Before you start this part of the assignment, it's worth spending 5-10 minutes quickly reviewing the specifications for Assignment 4 and your solution.

I've included some initial code to get you started in the starter file. Your class must contain:

- At least the following instance variables:
  - `self.movie` a string representing the title of the movie.
  - `self.current` a list of strings representing the current state of the game. The list will have one entry for each letter in the movie and will start out as all dashes. As the game is played, the dashes will get replaced by letters from the movie as the user guesses letters correctly.
  - `self.guessed` a list of all the letters guessed so far. It will start out empty.

  You may need additional instance variables and should feel free to add more. Do make sure that you only add instance variables in cases where you need to share information across methods. If the variable is only used inside one method it should be a local varial (i.e., no self.).

- A constructor that takes the movie title as input.
- A method `current_state_to_string` which takes zero arguments and returns a string representation of the current state of the game (this is similar to `list_to_string` from our original implementation).
- A method `guess` which takes a letter (as a string) as input and updates the state of the game as if the letter was guessed. Specifically, it should update the guessed letters and the current state of the game appropriately. You can either write this as a single method or as multiple methods. This method will likely have code from `insert_letter` and some from `play_hangman` from your original implementation.
- A method `has_won` that returns a boolean indicating whether or not the game has been won.
- A `__str__` method (already provided). This method relies on `self.guessed` being populated with the guessed letters correctly and on `current_state_to_string`. One difference between this implementation and the one from Assignment 4 is that we will not be printing out the guessed letters in a "pretty" way. Instead, we'll simply print it out in list form. Note, however, that it should still be in alphabetical order, so you'll need to make sure to keep the list sorted alphabetically.

To get a better feeling for how the class will be used, I've also provided a new version of the `play_hangman` function that uses the class above. Take a look at this function and make sure you understand how it works and how it relies on the different methods of the `Hangman` class.

Extra credit

To keep things simple, we justed printed out the guessed letters as a list. In our original version, we printed out a nicer version that didn't have the brackets and single quotes. For 1 point of extra credit, change the `__str__` method to return the guessed letters like our original version. You'll likely need to write a helper method to help you with this.

When you're done

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course, assignment number and the date.
- Each function should have an appropriate docstring.
- Each class should also have a docstring (right after the class definition) give a high-level description of the class.
- Include other miscellaneous comments to make things clear.

In addition, make sure that you've used good *style*. This includes:

- Following naming conventions, e.g. all variables and functions should be lowercase.
- Using good variable names.
- Proper use of whitespace, including indenting and use of blank lines to separate chunks of code that belong together.
- Make sure that none of the lines are too long.

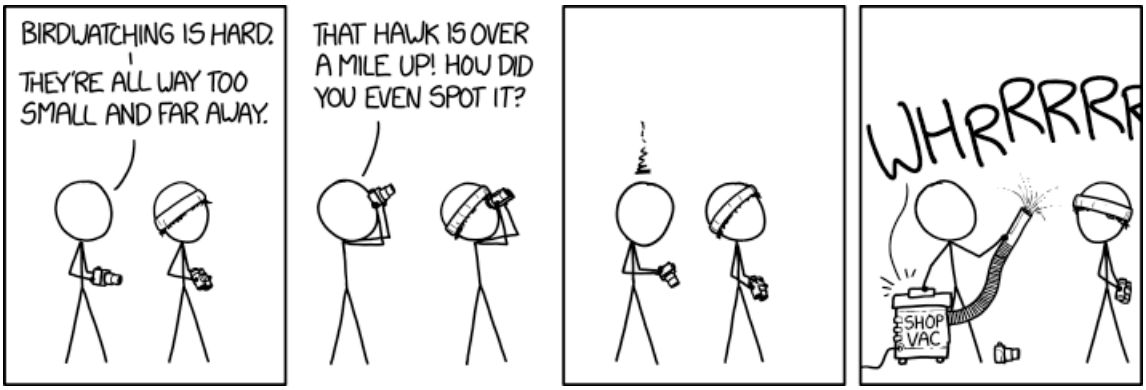
Submit your `assign8.py` file and ethics reading (in pdf format) online using the courses submission mechanism. *If you worked with a partner*, only one person should submit the assignment. Make sure that both of your names are at the top of your .py file and when submitting, select your partner during the submission process.

Ethics

Creating deepfakes is becoming easier and the results are more convincing. Read the article found in the starter (`deepfakes.pdf`), summarize it, and describe potential positive and negative uses of such technology.

Grading

	points
Warmup	5
constructor	2
`current_state_to_string`	2
`guess`	6
`has_won`	2
Comments, style	3
Ethics	1
extra credit	1
total	21 (+1)



http://xkcd.com/1826/