CS150 - Lab Prep 7
Due: Friday Oct. 28, at the beginning of class

For our lab, we're going to be writing a version of the word guessing game "hangman". For those who haven't played it before you can find many versions of it online or you can read more about it from Wikipedia (`http://en.wikipedia.org/wiki/Hangman_(game)`).

A demo

To get you familiar with what you'll be implementing for your assignment this week, I've provided you with an example program to run. Play with it a little bit before Friday. Note this will only work in the Mac lab[1]. To run it:

- Run `Terminal`. In your dock at the bottom of the screen there should be an icon that looks like:

  

  Click on it. If you can't find it in your dock, type `command + space_bar` to open the search in the upper right, then search for "Terminal" and select it that way.

- In the `Terminal` window type:

  ```
  python /home/dkauchak/PUBLIC/cs150/assignment6/hangman.pyc
  ```

  This should execute the example program in the terminal window. If you ever want to exit or quit early, you can type `control + c`.

You do not need to turn anything in for this section of the lab prep.

[1] If you're working from home or on your own laptop, send me an e-mail or stop by my office and I'll explain to you how to do it.

iterable

If you type:

```
>>> help(set)
```

and then look at the first few things it prints out, you'll see that the `set` class has two "constructors"

```
set() -> new empty set object
set(iterable) -> new set object
```

that is, two ways for creating new `set` objects. The first is used to create a new empty set. The second we've used with lists and strings to create new sets with some initial values.

The definition for this second constructor says that it takes as a parameter an object that is `iterable`. Some classes/objects are `iterable` and some are not. `iterable` classes/objects contain functionality that allow us to iterate over the elements in the list. For example, we can iterate over the items in an `iterable` object using a `for` loop:

```
for item in data:
    print item
```

`data` could be *any* `iterable` item. strings, lists and sets are all `iterable`, so we could assign any of these to `data` above and the loop would work (try it out if you're curious). Similarly, since the second constructor to `set` takes something that is `iterable` we could use any of these to create a new `set`. This should explains why when we create a set from a string, as in `set("abcd")` we get a set consisting of the four characters in the string and not the string itself.

Once you're comfortable with this idea, write a function called `iterable_to_string` that takes a single parameter, which is some `iterable` object, and returns a string consisting of each item in the `iterable` object converted to a string using `str` and concatenated together, separated by a space. For example, here are a few calls to this function:

```
>>> iterable_to_string("abcd")
'a b c d '
>>> iterable_to_string([4, 3, 2, 1])
'4 3 2 1 '
>>> iterable_to_string(set([4, 3, 2, 1]))
'1 2 3 4 '
```

Turn in your code for this function on a piece of paper with your name on it for you lab prep. This is the only thing you are required to turn in for lab prep this time.