6.189 Project 1

Readings

Get caught up on all the readings from this week!

What to hand in

Print out your hangman code and turn it in Monday, Jaunary 10 at 2:10 PM. Be sure to write *your name* and *section* clearly at the top of your code! You only have to pass in the Hangman code; you don't have to pass in answers to all the intermediate questions, but if you do them, you should save the code you write somewhere - you never know when written code might come in useful someday!

Project 1: The Game of Hangman

We're going to write the game of Hangman. This document provides a step-by-step approach to help you build the game. Use it as much or as little as you want. If you're uncertain, I recommend sticking with the document; however, if you want to try attacking this program on your own, that's great too. Actual coding starts in question 2.

1. Remember the maximum value trick we covered in lecture?[1]

Here's another problem along the same vein: Let's say I want to check and see if a number of facts are ALL true. For example, is every element in a list less than 6? Use what you learned from yesterday (and the homework) to write a *for* loop that will determine if all elements in a variable `some_list` are less than 6. Check your code using `some_list = [1,5,3,4]` and on `some_list = [5,3,7,5]`.

[1] If not, or we forgot to cover it, check this out...

```
def maxval(some_list):
    max_val = None
    for val in some_list:
        if val > max_val:
            max_val = val
    return max_val
```

We can also test if AT LEAST ONE fact is true. Write a *for* loop that will determine if at least one element in a list is less than 6. Test on `[7,8,7,9]` and `[7,2,5,8]`.

The first is the equivalent of checking A and B and C and D and ... The second is the equivalent of checking A or B or C or D or ...

2. Download the file `hangman_template.py` from the course website; save it as `hangman.py`. Also download `words.txt` and save it in the *same place*. We're going to start by storing the state of the game in variables at the top of the function `play_hangman`. The state is a complete description of all the information about the game. In the Nims game from homework 2, the state would be:

- The current player
- How many stones are in the pile

For Hangman, we need to store 3 pieces of information:

- `secret_word`: The word they are trying to guess (string).
- `letters_guessed`: The letters that they have guessed so far (list).
- `mistakes_made`: The number of incorrect guesses they've made so far (int).

You can name these something else if you'd like, but use a descriptive name. For now, set `secret_word` to be "claptrap". Once we've finished our program and got it working, then we'll change the `secret_word = 'claptrap'` to be `get_word()`, a function that pulls a random word from the file `words.txt`. This function is already defined for you. (This is called incremental programming - instead of trying to get everything right the first time, we'll get the basic program working then incrementally add small portions of code.) "claptrap" was selected because it's reasonably long and has duplicate letters – hopefully that will allow us to catch any bugs we might make.

Question – why can't we use `len(letters_guessed)` for `mistakes_made`?

Note the constant variable underneath the helper code:

```
MAX_GUESSES = 6
```

Constant just means that we won't change it. This isn't enforced by the compiler, so be careful not to accidentally change the value of `MAX_GUESSES`. My style is to put variables that I don't plan to change in all capital letters – other people do different things (some would have written `Max_guesses`, for example.) Any way works. We can decide what to do with this at the end (for example, should we have an "easy", "medium", "hard" mode with different numbers of guesses? As a programmer it's up to you to decide!)

Idea: At the end of the program, we should test our code with another word, one that has more than 6 distinct letters, to make sure that the program doesn't accidentally increment the number of mistakes on a correct guess.

3a. **Quickie reminder**: Enter the following lines of code in the prompt:

```
for i in "hello":
    print i
for i in ['a',True,123]:
    print i
```

Just a reminder on how *for* loops work.

3b. Let's start writing code! Here's our approach: we'll write functions to take care of smaller tasks that we need to do in hangman, then use them to write the actual game itself.

First, define the function `word_guessed()`. `word_guessed()` will return `True` if the player has successfully guessed the word, and `False` otherwise.

Example: If the `letters_guessed` variable has the value

`['a','l','m','c','e','t','r','p','n']`

`word_guessed()` will return `True`. If the `letters_guessed` variable has the value

`['e','l','q','t','r','p','n']`

`word_guessed()` will return `False`.

Hint: Obviously, you'll use a loop. There are two things you could loop over – the letters in `secret_word` or the letters in `letters_guessed`. Which one do you want to loop over? Don't just guess here, think! One of them makes sense and will be a lot easier than the other. You'll also be using the trick from the first problem.

4a. A quick break from Hangman to learn a little bit more about strings. Try this: type the following commands into the prompt.

```
dir()
a = 5
dir()
b = 3
c = 7
a = 14
dir()
from string import *
dir()
```

What does the dir function do? (Hint: `string` is a library - what does that mean?)

While still at the prompt, type `help(join)` and `help(lower)`.

4b. What lines of code belong in the missing spaces to achieve the desired outcome? (**Hint:** did you read about `join` and `lower` yet?)

```
>>> List1 = ['H','e','a','r']
>>> ???????
>>> ???????
>>> print string1
hear
```

5. Back to Hangman. So you'll want to use the string library. Note how we have added the line `from string import *` to the top of the template. This imports all of the functions from the `string` library, so you can use them as if you've defined them within your own file.

6. Now define the function `print_guessed()` that returns a string that contains the word with a dash '`-`' in place of letters not guessed yet.

Example: If the `letters_guessed` variable has the value `[]`, the expression `print print_guessed()` will print `--------`.

If the `letters_guessed` variable has the value `['a','p']`, the expression `print print_guessed()` will print `--ap--ap`.

If the `letters_guessed` variable has the value `['a','l','m','c','e','t','r','p','n']`, the expression `print print_guessed()` will print `claptrap`.

Hint: There are a lot of ways to go about this. One way is to iterate through `secret_word` and append the character you want to print to a list. Then use the join function to change the list into a string: your last line will look something like `return join(character_list, '')`

7. Now write the main game code. It helps to informally sketch out the code you want to write - this is called "pseudocode": an outline of what you are going to code that helps to guide you when you begin writing code. Here's an rough sketch of pseudocode, although you will want to expand on this:

```
continually loop:
  print ''n guesses left''
  print ''word''

  get letter in lowercase
  check - has letter already been guessed?
     If so, what should I do?
     If not, what should I do?
  check - is letter in word?
     If so, what should I do?
     If not, what should I do?
```

Write out some pseudocode that details what you want to do. It's a good idea to do this in comments within your code file, so you can use this as a guideline to write your code. **Hint:** remember to use the `break` statement if you use the continual loop!

8. Congratulations! You've finished the game. Now we want to make it look pretty so everyone else will be impressed as we are `:D`. Polish your game a bit using the following extensions:

1. Don't use the word "claptrap" every time! Underneath the function `play_hangman` you should see a commented line that looks like this:

   ```
   # secret_word  = get_word()
   ```

   Remove the '`#`' before it, and the secret word will be a new, random word each time!

2. *Optional*: ASCII GRAPHICS! On the `materials` section of the course webpage, you can find two files called `hangman_lib.py` and `hangman_lib_demo.py`. The first contains a set of ASCII graphics that you can use in your code; the second shows how to use the package. You can insert these into your Hangman game to make it much more exciting than it was ;)

   Hint: Remember to add the line `from hangman_lib import *` at the top of your code, just like in `hangman_lib_demo.py`. *Do not copy the graphics into your code!* Just use the import statement!

3. *Optional*: Allow the user the option of guessing the full word early (perhaps by modifying your prompt to say something like, `Enter a letter, or the word 'guess' to try and guess the full word:`) Then, allow the user a try to enter in the full word) You may want to take off 2 guesses if they enter an incorrect word...

4. *Optional*: Modify your `print_guessed()` function such that, in addition to what it already prints out, it prints out the letters the user has **not** yet guessed.

On Monday turn in the code for your completed Hangman game, stapled together, with your name on top in comments.

6.189 A Gentle Introduction to Programming

January IAP 2011