Problem Set 3:

Introduction:

Note: Do not be intimidated by this problem! It's actually easier than it looks. We will 'scaffold' this problem, guiding you through the creation of helper functions before you implement the actual game.

For this problem, you will implement a variation of the classic wordgame Hangman. For those of you who are unfamiliar with the rules, you may read all about it here. In this problem, the second player will always be the computer, who will be picking a word at random.

In this problem, you will implement a **function**, called `hangman`, that will start up and carry out an interactive Hangman game between a player and the computer. Before we get to this function, we'll first implement a few helper functions to get you going.

The code we have given you loads in a list of words from a file (or a default). If everything is working okay, after a small delay, you should see the following printed out:

```
Loading word list from file...
55909 words loaded.
```

The file `pset3.py` has a number of already implemented functions you can use while writing up your solution. You can ignore the code between the following comments, though you should read and understand how to use each helper function by reading the docstrings.

You will want to do all of your coding for this problem within this file as well because you will be writing a program that depends on each function you write.

Requirements

Here are the requirements for your game:

1. The computer must select a word at random from the list of available words that was provided in `words.txt`. The functions for loading the word list and selecting a random word have already been provided for you in `pset3.py`.
2. The game must be interactive; the flow of the game should go as follows:
    - At the start of the game, let the user know how many letters the computer's word contains.
    - Ask the user to supply one guess (i.e. letter) per round.
    - The user should receive feedback immediately after each guess about whether their guess appears in the computer's word.
    - After each round, you should also display to the user the partially guessed word so far, as well as letters that the user has not yet guessed.
3. Some additional rules of the game:
    - A user is allowed 8 guesses. Make sure to remind the user of how many guesses s/he has left after each round. Assume that players will only ever submit one character at a time (A-Z).
    - A user loses a guess **only** when s/he guesses incorrectly.
    - If the user guesses the same letter twice, do not take away a guess - instead, print a message letting them know they've already guessed that letter and ask them to try again.
    - The game should end when the user constructs the full word or runs out of guesses. If the player runs out of guesses (s/he "loses"), reveal the word to the user when the game ends.

Advice:

- The code runs the test cases when you call it like `python pset3.py`.
- If you want to test a particular function with a particular input, you can do:

```
Python
>>> from pset3 import *
>>> isWordGuessed(“secret”, “attempt”)
False
```

- If you want to to run the hangman game interactively, you can do:
  ```
  Python
  >>> from pset3 import *
  >>> play_hangman_demo()
  [... game ...]
  ```

Sample Output

The output of a winning game should look like this...

```
Loading word list from file...
55900 words loaded.
Welcome to the game, Hangman!
I am thinking of a word that is 4 letters long.
-------------
You have 8 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: _ a_ _
------------
You have 8 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! You've already guessed that letter: _ a_ _
------------
You have 8 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: s
Oops! That letter is not in my word: _ a_ _
------------
You have 7 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: t
Good guess: ta_ t
------------
You have 7 guesses left.
Available letters: bcdefghijklmnopqruvwxyz
Please guess a letter: r
Oops! That letter is not in my word: ta_ t
------------
You have 6 guesses left.
```

```
Available letters: bcdefghijklmnopquvwxyz
Please guess a letter: m
Oops! That letter is not in my word: ta_ t
------------
You have 5 guesses left.
Available letters: bcdefghijklnopquvwxyz
Please guess a letter: c
Good guess: tact
------------
Congratulations, you won!
```

And the output of a losing game should look like this...

```
Loading word list from file...
55900 words loaded.
Welcome to the game Hangman!
I am thinking of a word that is 4 letters long
-----------
You have 8 guesses left
Available Letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! That letter is not in my word: _ _ _ _
-----------
You have 7 guesses left
Available Letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: _ _ _ _
-----------
You have 6 guesses left
Available Letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: c
Oops! That letter is not in my word: _ _ _ _
-----------
You have 5 guesses left
Available Letters: defghijklmnopqrstuvwxyz
Please guess a letter: d
Oops! That letter is not in my word: _ _ _ _
-----------
You have 4 guesses left
Available Letters: efghijklmnopqrstuvwxyz
```

```
Please guess a letter: e
Good guess: e_ _ e
-----------
You have 4 guesses left
Available Letters: fghijklmnopqrstuvwxyz
Please guess a letter: f
Oops! That letter is not in my word: e_ _ e
-----------
You have 3 guesses left
Available Letters: ghijklmnopqrstuvwxyz
Please guess a letter: g
Oops! That letter is not in my word: e_ _ e
-----------
You have 2 guesses left
Available Letters: hijklmnopqrstuvwxyz
Please guess a letter: h
Oops! That letter is not in my word: e_ _ e
-----------
You have 1 guesses left
Available Letters: ijklmnopqrstuvwxyz
Please guess a letter: i
Oops! That letter is not in my word: e_ _ e
-----------
Sorry, you ran out of guesses. The word was else.
```

On the next page, we'll break down the problem into logical subtasks, creating helper functions you will need to have in order for this game to work.

Problem 1 - Is the Word Guessed

Please read the Hangman Introduction before starting this problem. We'll start by writing 3 simple functions that will help us easily code the Hangman problem. First, implement the function `isWordGuessed` that takes in two parameters - a string, `secretWord`, and a list of letters, `lettersGuessed`. This function returns a boolean - `True` if `secretWord` has been guessed (ie, all the letters of `secretWord` are in `lettersGuessed`) and `False` otherwise.

Example Usage:

```
>>> secretWord = 'apple'

>>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']

>>> print(isWordGuessed(secretWord, lettersGuessed))

False
```

For this function, you may assume that all the letters in `secretWord` and `lettersGuessed` are lowercase.

Problem 2 - Getting the User's Guess

Next, implement the function `getGuessedWord` that takes in two parameters - a string, `secretWord`, and a list of letters, `lettersGuessed`. This function returns a string that is comprised of letters and underscores, based on what letters in `lettersGuessed` are in `secretWord`. This shouldn't be too different from `isWordGuessed`!

Example Usage:

```
>>> secretWord = 'apple'
>>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']
>>> print(getGuessedWord(secretWord, lettersGuessed))
'_ pp_ e'
```

When inserting underscores into your string, it's a good idea to add at least a space after each one, so it's clear to the user how many unguessed letters are left in the string (compare the readability of `____` with `_ _ _ _ `). This is called *usability* - it's very important, when programming, to consider the usability of your program. If users find your program difficult to understand or operate, they won't use it!

For this problem, you are free to use spacing in any way you wish - our grader will only check that the letters and underscores are in the proper order; it will not look at spacing. We do encourage you to think about usability when designing.

For this function, you may assume that all the letters in `secretWord` and `lettersGuessed` are lowercase.

Problem 3 - Printing Out all Available Letters

Next, implement the function `getAvailableLetters` that takes in one parameter - a list of letters, `lettersGuessed`. This function returns a string that is comprised of lowercase English letters - all lowercase English letters that are **not** in `lettersGuessed`.

Example Usage:

```
>>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']
>>> print(getAvailableLetters(lettersGuessed))
abcdfghjlmnoqtuvwxyz
```

Note that this function should return the letters in alphabetical order, as in the example above.

For this function, you may assume that all the letters in `lettersGuessed` are lowercase.

Hint: You might consider using `string.ascii_lowercase`, which is a string comprised of all lowercase letters:

```
>>> import string
>>> print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz
```

Problem 4 - The Game

Now you will implement the function `hangman`, which takes one parameter - the `secretWord` the user is to guess. This starts up an interactive game of Hangman between the user and the computer. Be sure you take advantage of the three helper functions, `isWordGuessed`, `getGuessedWord`, and `getAvailableLetters`, that you've defined in the previous part.

Hints:

- You should start by noticing where we're using the provided functions (at the top of `pset3.py`) to load the words and pick a random one. Note that the functions `loadWords` and `chooseWord` should only be used on your local machine, not in the tutor. When you enter in your solution in the tutor, you only need to give your `hangman` function.

Consider using `lower()` to convert user input to lower case. For example:
guess = 'A'

- guessInLowerCase = guess.lower()
- Consider writing additional helper functions if you need them!
- There are four important pieces of information you may wish to store:
    1. `secretWord`: The word to guess.
    2. `lettersGuessed`: The letters that have been guessed so far.
    3. `mistakesMade`: The number of incorrect guesses made so far.
    4. `availableLetters`: The letters that may still be guessed. Every time a player guesses a letter, the guessed letter must be removed from `availableLetters` (and if they guess a letter that is not in `availableLetters`, you should print a message telling them they've already guessed that - so try again!).

Sample Output

The output of a winning game should look like this...

```
Loading word list from file...
        55900 words loaded.
        Welcome to the game Hangman!
        I am thinking of a word that is 4 letters long.
        -------------
        You have 8 guesses left.
        Available letters: abcdefghijklmnopqrstuvwxyz
        Please guess a letter: a
        Good guess: _ a_ _
        ------------
        You have 8 guesses left.
        Available letters: bcdefghijklmnopqrstuvwxyz
        Please guess a letter: a
        Oops! You've already guessed that letter: _ a_ _
        ------------
        You have 8 guesses left.
        Available letters: bcdefghijklmnopqrstuvwxyz
        Please guess a letter: s
        Oops! That letter is not in my word: _ a_ _
        ------------
        You have 7 guesses left.
        Available letters: bcdefghijklmnopqrtuvwxyz
        Please guess a letter: t
        Good guess: ta_ t
```

```
------------
You have 7 guesses left.
Available letters: bcdefghijklmnopqruvwxyz
Please guess a letter: r
Oops! That letter is not in my word: ta_ t
------------
You have 6 guesses left.
Available letters: bcdefghijklmnopquvwxyz
Please guess a letter: m
Oops! That letter is not in my word: ta_ t
------------
You have 5 guesses left.
Available letters: bcdefghijklnopquvwxyz
Please guess a letter: c
Good guess: tact
------------
Congratulations, you won!
```

And the output of a losing game should look like this...

```
Loading word list from file...
55900 words loaded.
Welcome to the game Hangman!
I am thinking of a word that is 4 letters long.
-----------
You have 8 guesses left.
Available Letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! That letter is not in my word: _ _ _ _
-----------
You have 7 guesses left.
Available Letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: _ _ _ _
-----------
You have 6 guesses left.
Available Letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: c
Oops! That letter is not in my word: _ _ _ _
-----------
You have 5 guesses left.
Available Letters: defghijklmnopqrstuvwxyz
Please guess a letter: d
Oops! That letter is not in my word: _ _ _ _
-----------
You have 4 guesses left.
Available Letters: efghijklmnopqrstuvwxyz
Please guess a letter: e
Good guess: e_ _ e
```

```
        -----------
        You have 4 guesses left.
        Available Letters: fghijklmnopqrstuvwxyz
        Please guess a letter: f
        Oops! That letter is not in my word: e_ _ e
        -----------
        You have 3 guesses left.
        Available Letters: ghijklmnopqrstuvwxyz
        Please guess a letter: g
        Oops! That letter is not in my word: e_ _ e
        -----------
        You have 2 guesses left.
        Available Letters: hijklmnopqrstuvwxyz
        Please guess a letter: h
        Oops! That letter is not in my word: e_ _ e
        -----------
        You have 1 guesses left.
        Available Letters: ijklmnopqrstuvwxyz
        Please guess a letter: i
        Oops! That letter is not in my word: e_ _ e
        -----------
        Sorry, you ran out of guesses. The word was else.
```

Note that if you choose to use the helper functions `isWordGuessed`, `getGuessedWord`, or `getAvailableLetters`, you do not need to paste your definitions in the box. We have supplied our implementations of these functions for your use in this part of the problem. If you use additional helper functions, you will need to paste those definitions here.

Your function should include calls to `input` to get the user's guess.

Why does my Output Have `None` at Various Places?

`None` is a keyword and it comes from the fact that you are printing the result of a function that does not return anything. For example:

```
  def foo(x):
      print(x)
```

If you just call the function with `foo(3)`, you will see output:

```
  3   #-- because the function printed the variable
```

However, if you do `print(foo(3))`, you will see output:

```
  3    #-- because the function printed the variable
  None  #-- because you printed the function (and hence the return)
```

All functions return something. If a function you write does not return anything (and just prints something to the console), then the default action in Python is to `return None`