Instructions

This is a time bound assignment, please complete and submit it within 3 hours of receiving the assignment.

If you are not familiar with Hangman, please familiarize thoroughly with the game before proceeding. Its a game where the user/player needs to guess a phrase generated by the program, by guessing one letter at a time. The player only gets a fixed number of incorrect guesses before he loses [Wiki].

While Hangman is typically a very visual game, our code implementation will **not** incorporate any graphics or UI. You are free to use your language of choice.

Part 1

Before beginning to program the game, you need to curate/create a list of english phrases. This will serve as your "Phrase Bank" from which you will randomly select a phrase for serving the game to the user in Part 2 of the assignment. All phrases in the phrase bank must adhere to the following rules:

- Number of phrases should at least be 10K+.
- Phrases should have 2 - 3 words (min 2 words, maximum 3 words).
- Phrases could be any 2-3 consecutive words (space separated) that appear in an english sentence. e.g. in a sentence like 'I like all kinds of ice cream. What do you like?', you could generate phrases like 'I like', 'kinds of', 'ice cream'. All these are valid phrases. Words separated by punctuations or anything other than spaces, won't be considered a valid phrase e.g. **'cream. What'** is **not** a valid phrase. Similarly, **'I cream'** is **not** a valid phrase since the two words don't appear consecutively in the sentence/corpus.
- Phrases should only contain letters [a-zA-Z] and spaces and no other characters.
- You shouldn't modify the strings from the corpus in any way e.g. by removing punctuations, or by removing certain words to adhere to the above rules. The phrases should be extracted as is from the english sentence corpus without any modifications - if the phrase doesn't fit the rules, it should be discarded rather than adjusted to adhere to the rules.
- Here is the process to create the phrase bank:
    - Find a reasonably big english sentence/essay data/corpus from the internet. It doesn't need to be a huge corpus - it should be big enough so that you can generate around 10K phrases.
    - Write a script to parse the data corpus, and extract reasonably random (there could be unintentional partial duplication, but the phrases should more or less be unique/random) phrases from the corpus following the above conditions.

Getting the 10K+ phrases adhering to the rules within the allotted time **is more important** than efficiency or great code.

For Part 1 submission, please document and submit the following:

- Reference/Link to the sentence corpus used
- Script written to parse/extract phrases from the corpus
- Actual phrase bank generated

Part 2

When programming the game for the user, the program will select one phrase randomly from the curated phrase bank. Once a phrase is selected, you should show the underscores "_" (one for each letter) and repeatedly ask the user for a letter until either they guess the word, or they run out of incorrect guesses. The user should have exactly 6 incorrect guesses , but correctly guessing a letter should not count as an incorrect guess.

After every guess, the user should be able to see:

- The blank letters in the phrase, with correctly-guessed letters filled in
- Their correctly guessed letters
- Their incorrectly guessed letters
- How many incorrect guesses they have left

To see an example, please scroll down to *Sample Output*. It does not have to match exactly, but it should serve as a general basis of what information the user should have access to as they make their guesses.

You as the game developer should also make sure to count for bad user inputs, same incorrect guesses and other edge cases making sure the game experience is as coherent as possible, and handling most of the edge cases to provide a smooth experience.

Evaluation criteria:

Since this is a time bound assignment, it is important to manage your time efficiently so that you can complete the whole assignment rather than spending too much time on any one part.

- **Correctness:** Handling edge cases, bug free code, that works end to end is most important**.**
  e.g.:
  - Handle all edge cases such that even if the user input is wrong, the game handles it correctly
  - Handle all edge cases such as handling same wrong character inputs, case-insensitive phrases/guesses, invalid characters, etc.

- **Code Quality:** Code quality and readability will be one of the most important evaluation criteria. Here are a few general things to keep in mind:
    - Use functions to break your problem down into smaller, more manageable pieces. If you have a function that is doing multiple things, then break it down into multiple functions.
    - Think deeply about naming conventions and names that you use throughout the code
    - Think about how to organize the code within abstractions, within files, within methods and try to do it with a very high quality
    - Readability should be very high - the code should be intuitively programmed, comments should be added wherever its hard to understand the logic, names should be intuitive - a completely new developer should be able to grasp and understand the code with minimal oversight
- **Documentation:** Please document instructions on how to execute/test the program

Sample output

Note this is just a sample output and your actual output might look significantly different from the below output as you handle all the edge cases possible while simulating the game.

```
_ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _
Guess a letter: a

Nope!
_ _ _ _ _ _ _ _   _ _ _ _ _ _ _ _
Correct letters: []
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: e

Good!
_ _ _ _ _ _ e _   _ _ _ e _ _ e
Correct letters: ['e']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: i

Good!
_ _ _ _ _ _ e _   _ _ i e _ _ e
Correct letters: ['e', 'i']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: o

Good!
```

```
_ o _ _ _ _ e _    _ _ i e _ _ e
Correct letters: ['e', 'i', 'o']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: u

Good!
_ o _ _ u _ e _    _ _ i e _ _ e
Correct letters: ['e', 'i', 'o', 'u']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: t

Good!
_ o _ _ u t e _    _ _ i e _ _ e
Correct letters: ['e', 'i', 'o', 'u', 't']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: m

Good!
_ o m _ u t e _    _ _ i e _ _ e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm']
Incorrect letters: ['a']
Chances: 1/6
Guess a letter: h

Nope!
_ o m _ u t e _    _ _ i e _ _ e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm']
Incorrect letters: ['a', 'h']
Chances: 2/6
Guess a letter: s

Good!
_ o m _ u t e _    s _ i e _ _ e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm', 's']
Incorrect letters: ['a', 'h']
Chances: 2/6
Guess a letter: c

Good!
c o m _ u t e _    s c i e _ c e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm', 's', 'c']
Incorrect letters: ['a', 'h']
Chances: 2/6
Guess a letter: p
```

```
Good!
c o m p u t e _   s c i e _ c e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm', 's', 'c', 'p']
Incorrect letters: ['a', 'h']
Chances: 2/6
Guess a letter: r

Good!
c o m p u t e r   s c i e _ c e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm', 's', 'c', 'p', 'r']
Incorrect letters: ['a', 'h']
Chances: 2/6
Guess a letter: n

Good!
c o m p u t e r   s c i e n c e
Correct letters: ['e', 'i', 'o', 'u', 't', 'm', 's', 'c', 'p', 'r', 'n']
Incorrect letters: ['a', 'h']
Chances: 2/6

You got it!  Great job!
```

Submission instructions

Please email the code to nikhil@gammanet.com.