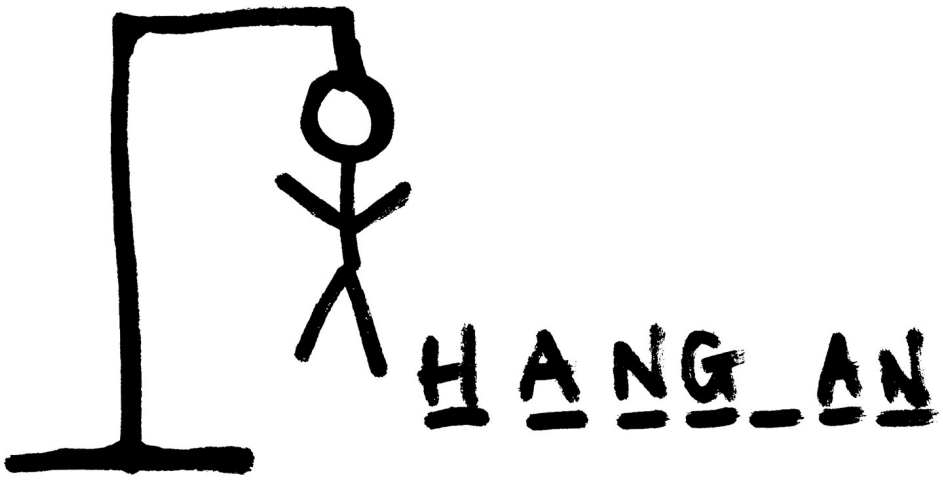
The Ultimate Hangman Game



Task: You will make an all-too-familiar Hangman Game in this final course project :). (If you don't know what the Hangman game is, you are probably able to get a hint for it from the picture above.) This game will have lots of interactions with the user, through the console.

Requirements:

Preset

1. A word dictionary. You will have a list that stores a "word dictionary", of **No Less Than** 10 words (preferably more). You will hardcode in this list in your gameplay, and so the list will be created when the game is run. **All words should be lowercase.**
2. With each word in the word dictionary, there will be a "hint" sentence that is displayed when this word is chosen. This means that the best data structure to store your word & hint dictionary is a **tuple** of **(word, hint)**. Your word dictionary should be a list of tuples.
3. Note: Your hangman game, ideally, should have a theme, and your word dictionary should have words revolving around that theme. For example, I love to ski, so my game will be "Hangman at the Great Canadian Ski Resorts" and all my words will be related to skiing.
4. When the game is run, your game will start by greeting the user with "Welcome to [name of your hangman game, give it a cool name]!" You can ask the user for their name, other information, etc. This is up to you. You want to make it pleasant for the player.

5. For the Welcome screen, I'll be really impressed if you put up a nice ASCII art ;) - Not a requirement though.

Gameplay

1. Once you've gotten enough information from the user, and have started the game. "Starting [name of your game] for [user name], good luck!" Your game will display the instruction as follows:
    a. There are 5 rounds in this hangman game.
    b. You will win the game once you finish all 5 rounds.
    c. Each round will consist of a single word from our dictionary, and you will guess the word.
    d. Each letter that you guess that is **NOT** in the word, will add a stroke to the little guy → see right. **The little stick man will have 6 lives. Namely: Head, body, left arm, right arm, left leg, right leg.**
    e. Once the little stick man is entirely "hung", you will have exhausted your guess chances, and it's game over, you lost.
    f. If you guessed the word without the little stick man being hung, you have won the round and will move onto the next round.
    g. Once you complete all 5 rounds, you are declared a winner.
2. Let the user know "Starting round X…"
3. Use the random number generator, choose a random word from your list of words. Print the following on the screen: The hint to the word, the hangman structure, and dashes "-" to represent the number of letters in the word. Note that space should just be represented with a space. Such as:

```
This is the largest ski resort in Ontario, both in terms of visitors,
number of runs, and altitude. It's about 2 hours north of Toronto.
 |----
 |
 |
 |
---
Your guess so far: ---- --------
Incorrect letters:
```

```
Guess a letter:
```

4. Now the user will enter a letter. You will need to take care of bad inputs. That is, only accept if the user enters **lower case letters a-z.** Reject everything else and ask them to re-enter. (Note that it's easier to only allow the valid and reject everything else, than checking invalidity of user input).
5. If the user guesses a letter **correctly**, say, from the above example, I guessed "u". Your program should display:

```
This is the largest ski resort in Ontario, both in terms of visitors,
number of runs, and altitude. It's about 2 hours north of Toronto."
 |----
 |
 |
 |
---
Your guess so far: --u- --u-----
Incorrect letters:

Guess a letter:
```

6. If the user guesses a letter **incorrectly**, say, from the above example, I guessed "y". Your program should display:

```
This is the largest ski resort in Ontario, both in terms of visitors,
number of runs, and altitude. It's about 2 hours north of Toronto.
 |----
 |   O
 |
 |
---
Your guess so far: --u- --u-----
Incorrect letters: y

Guess a letter:
```

Notice how the little stick man got a head.

7. This game play will continue until either:
    a. The stick man has grown all 6 parts, OR
    b. The user guessed the full word without the stick man getting hung.

LOST - Stick man grew all 6 parts:

```
This is the largest ski resort in Ontario, both in terms of visitors,
number of runs, and altitude. It's about 2 hours north of Toronto.
 |----
 |   O
 |  \|/
 |  / \
---
Your guess so far: --u- --u-----
Incorrect letters: y x z r w p

OOPS, you've exhausted your chances. Game over, better luck next
time!
```

ONTO THE NEXT ROUND - User guessed full word:

```
This is the largest ski resort in Ontario, both in terms of visitors,
number of runs, and altitude. It's about 2 hours north of Toronto.
 |----
 |   O
 |  \|
 |
---
Your guess so far: blue mountain
Incorrect letters: y x z

Look! You got it! Moving on to the next round...
```

8. If the user has finished all 5 rounds then you should display a congratulatory end game screen.

Hints for Implementation - READ THIS - This tells you exactly what to do:

1. Choosing a random word for each round - use a random number generator and choose the (word, hint) tuple at that index in your word list.
2. Keeping track of incorrect letters:
    a. Create an empty list at the start of each round (i.e. set the variable to **incorrect_letters = []** at the start of each round).
    b. Check if the user's guess is in the word string using "in".
    c. As the user guesses each incorrect letter, append the incorrect letter to the incorrect_letters list.
3. Outputting "-" in place of letters:

    a. Create a new empty string.
    b. Use a for loop to go through your word string, everywhere that it has a letter, + a "-" to your new string. Output the new "-" string.
4. Checking if the user's guess is a letter contained in the word and replacing "-" with the correct letter:
    a. Convert your "-" string to a list.
    b. Then, you should iterate through the word string using a for loop. For each index that the user input matches the letter in your word, go to that **same index** in your "-" list and replace the "-" in that index with the correct letter.
    c. Once the for loops terminates, convert the "-" back to a string and print it to screen. This should get you the "-" string with the correct letters replaced by user input.
5. Drawing the stick man:
    a. The easiest way to do this is to have a list of 7 strings, each representing a phase of the stick man (the first one is the picture of the stand without the stick man yet), call it **stickman_art**.
    b. The strings should be saved like "|---\n|\n|\n|\n---" for the base. (You can make your own for the little stick guy). Try print it out to see how it looks before finalizing.
    c. At the beginning of each round, print out the stickman picture at index [0] of this **stickman_art**.
    d. As the user enters each incorrect letter, print out the stickman picture at index [i] of **stickman_art** where i is the number of incorrect letters.
6. How to know the word has been fully guessed.
    a. (Method 1) Go through your "-" string. If there are no more "-", then the word is done.
    b. (Method 2) Do a string comparison. If **the user guess string is == the word string**, then the word is done.

*****IF you are truly finding this too challenging, you may do a "reduced" version of this game:**

- Only 1 round, instead of 5
- No need to draw stick man, simply output how many tries the user has left - the user still gets 6 tries.
- Do not need to take care of bad user input during game play - you may assume the user will always enter lower case letters a-z.
- Still need to keep an incorrect letters list
- Still need to replace the "-" string with each correct letter the user inputs.
- Let the user know if they won or lost.

If you choose to do the reduced version, the maximum you can get on the project is 75%.