Assignment 4: Hangman

(Guess the Word)

CompSci 101 Fall 2021

Due: November 2, 2021

Overview

You must create a playable Hangman game that allows the user to try and guess a secret word. See the description below for more specific requirements.

This assignment provides specifications for certain functions because it facilitates the process of debugging, grading, and providing feedback. However, you are given some freedom with regard to how these functions should fit together into a working Hangman game. In other words, these instructions provide a framework for completing the assignment, but **part of the process of creating this Hangman game will involve figuring out how to use this framework**. Moreover, this is your chance to develop your skills on how to organize and write code. This skill takes time and practice to develop, and this assignment is an opportunity for that practice. You should feel free to discuss with your peers and TAs on what is the "right way" to do things, while also keeping in mind there is more than one way to solve this assignment.

You'll need to use the starter code from this zip file and create a project from it. Download the zip file, unzip the file, then go into Pycharm and select *File* > *Open…* then select the unzipped folder. Choose either New Window or This Window.The starter code should have the following:

1. **`Hangman.py`** - a Python file containing the starter code for this assignment
2. **`lowerwords.txt`** - a text file containing words to be used in the game
3. **`CheckMyFunctions.py`** - a Python helper file for testing your Hangman functions

Learning Goals

1. Implementing functions based on a specification.
2. Practicing putting together many functions to create complex behavior.
3. Using while loops.
4. Handling lots of user input.
5. Practice at string manipulation.

Completing the Assignment

You are given the starter code you will need to complete this assignment. This code is described in detail in the section Functions To Implement. **For full credit, you should have no global variables and use all the functions as described.**

Requirements

1. The user must be given a choice of (e)asy or (h)ard mode, which corresponds to 12 and 8 misses (incorrect guesses) until hung, respectively. You ***do not need*** to check for bad user-input; you can assume the user will enter an 'h' or an 'e'.
    a. In the runs below, the default is 12 (easy) and the user can enter 'h' for hard.
2. A secret word must be chosen at random from the file **`lowerwords.txt`**. First, a length between 5 and 10 (inclusive) ***must be chosen at random***, and then a word from all the words of that length ***must be chosen at random*** as the secret word.

3. The user must be ***shown all the letters already guessed***, and these letters must appear in ***sorted order*** (see the runs below). These include letters in the word as well as missed letters. Use the Python **`sorted(...)`** function to do this.
4. The user must be told how many misses are left on each turn (when prompted to enter a letter). When a letter in the word is correctly guessed, the number of misses left doesn't change, but an incorrectly-guessed letter changes the number of misses left.
5. If the user repeats a previous letter guess, this should not count as a miss and should not count as a guess when reporting summary game statistics.
6. When either the user ***has guessed the word*** or the number of ***misses allowed is reached***, the game should be over and *individual game summary statistics* about the number of guesses and misses should be printed. See the sample runs below.
7. On each turn, the secret "word" should be printed with ***underscores for unknown letters*** and correctly-guessed letters in place as warranted. A single space should appear between each underscore and letter.
8. After each Hangman game, the user should be prompted to play again. The user can play as many games as they want. Once they decide to quit, *summary statistics for the entire "session" (multiple games)* should be printed, showing the total win-loss count.
    a. You should implement this functionality in the main program block.

Functions to Implement

`handleUserInputDifficulty()`

This function first prints "**`How many misses do you want? Hard has 8 and Easy has 12`**." Then, it uses the **`input(...)`** function to print "**`(h)ard or (e)asy>`** " (note the space after the ">") and take as input from the user a string of either "**`e`**" or "**`h`**".

Parameters:
No parameters

Returns:
Type: int
The number of allowed misses for the game, so 12 if user inputs 'e' and 8 for 'h'.

Assumptions:
The user only inputs 'e' or 'h', input checking is not necessary.

`getWord(words, length)`

Randomly picks a word from **`words`** that has **`length`** number of characters in it.

Parameters:

- words (type: list of strings) - potential words to use for the hangman game, which will be sourced from the **`lowerwords.txt`** file in **`runGame()`**
- length (type: int) - the length of the word to output

Returns:
Type: string
The word to use for this hangman game, which is a randomly chosen string in the list **`words`** that is of length **`length`**.

`createDisplayString(lettersGuessed, missesLeft, hangmanWord)`

Creates the string that will be displayed to the user, using the information in the parameters. ***Ensure that the display string contains spaces between characters in the right places as shown in the example below.***

Parameters:
- lettersGuessed (type: list of strings) - a list of one character strings where each is a letter that the user has already guessed
- missesLeft (type: int) - the number of incorrect guesses the user has remaining
- hangmanWord (type: list of strings) - represents the currently displayed hangman, each element in the list represents a letter in the secret word and it is either the actual letter or "_" representing that the user has not yet guessed that letter

Returns:
Type: string
A string that should be displayed to the user at each turn. Like so:

```
letters you've guessed: SPACE_SEPARATED_STR_OF_LETTERSGUESSED
misses remaining = MISSESLEFT
SPACE_SEPARATED_STR_OF_HANGMANWORD
```

Example:

```
letters you've guessed: a e i o s u
misses remaining = 4
s _ _ _ o o _
```

`handleUserInputLetterGuess(lettersGuessed, displayString)`

Handles the user inputting a letter to guess. It first prints out the displayString, then calls the input function with "**`letter>`** " (note the space after ">") and takes an input from the user. If that input is not in lettersGuessed, it returns that value. If that input is in lettersGuessed, it prints out "**`you already guessed that`**" and starts over. Therefore, this function should not return

until the user inputs a valid character that has not been guessed yet. We recommend using a while loop to handle this (see the note about while loops below if your program gets into an infinite loop.)

Parameters:

- lettersGuessed (type: list of strings) - a list of one character strings where each is a letter that the user has already guessed
- displayString (type: string) - the value returned by createDisplayString

Returns:
Type: string
a string of length one that the user has inputted and is not in the list lettersGuessed

Assumptions:
The user will input a string of length 1.

`updateHangmanWord(guessedLetter, secretWord, hangmanWord)`

If guessedLetter is in the secretWord, it updates the corresponding positions in the list hangmanWord with that letter. This function should handle the case where guessedLetter is not in secretWord.

Parameters:

- guessedLetter (type: string) - the user's current guess, represented as a string of length 1, the return value of handleUserInputLetterGuess
- secretWord (type: string) - the secret word returned by getWord
- hangmanWord (type: list of strings) - represents the currently displayed hangman, each element in the list represents a letter in the secret word and it is either the actual letter or "_" representing that the user has not yet guessed that letter

Returns:
Type: list
The new hangmanWord, which is a list of strings where each string is a single letter either corresponding to the same letter in secretWord or '_' if the user has not guessed the letter yet in the game.

Example:

```
guessedLetter = "a"
secretWord = "cat"
hangmanWord = ["c", "_", "_"]
Returns ["c", "a", "_"]
```

`processUserGuess(guessedLetter, secretWord, hangmanWord, missesLeft)`

Takes the user's guess, the secret word, the user's current progress on the word, and the number of misses left and updates the current game state of hangmanWord, number of misses left, and whether the user missed.

Parameters:

- guessedLetter (type: string) - the user's current guess, represented as a string of length 1, the return value of handleUserInputLetterGuess
- secretWord (type: string) - the secret word returned by getWord
- hangmanWord (type: list of strings) - represents the currently displayed hangman, each element in the list represents a letter in the secret word and it is either the actual letter or "_" representing that the user has not yet guessed that letter
- missesLeft (type: int) - the number of misses the user has left

Returns:

Type: list
A list with the following at each index:

- Index 0: (type: list of strings) the "new" hangmanWord based on the user's guess in guessedLetter, it should use the helper function updateHangmanWord to do this.
- Index 1: (type: int) an updated value for missesLeft based on the user's guess in guessedLetter
- Index 2: (type: bool) indication of whether the user made a correct guess, where True means the user guessed a letter in the word and False means the user missed

Assumptions:

guessedLetter has the value returned from handleUserInputLetterGuess, which means that it is guaranteed to be a letter that the user has not already guessed. hangmanWord is the same length as secretWord and any letter in hangmanWord that is not '_' is the corresponding letter in secretWord.

`runGame(filename)`

This is the function called in the main block. It does the following:

1. Using the helper functions, it runs the game.
2. Setup the game
    a. Loads the words from the file at the location specified in the parameter filename.
    b. Randomly chooses the length of the secret word between 5-10 inclusive.
    c. Calls handleUserInputDifficulty and getWord
    d. After calling the function getWord, it creates the list of strings that holds the currently displayed hangman (a.k.a. hangmanWord).

3. After the initial setup of the game, it should use a loop (we highly recommend the while loop - see note about while loops) to run each round of the game (where “round” means each time the user guesses a letter). In this loop it does the following:
    a. Tracks: the letters guessed, the current hangmanWord, the number of misses left
    b. Calls createDisplayString, handleUserInputLetterGuess, and processUserGuess.
    c. Prints out whether the currently guessed letter is in the word.
    d. Determines whether the game is over, where the user:
        i. Won if there are no more "_" in hangmanWord
        ii. Lost if missesLeft <= 0
4. Prints the ending message to the user.
5. When the game is over, returns whether the user won.

Parameters:
filename (string) - string with the name of the file (including file extension) containing the words to use for the game. This file can be found in the downloaded zip file described in the Overview.

Prints:
1. During the game it prints out whether the currently guessed letter is in the word by printing "**`you missed: LETTER not in word`**", where LETTER is replaced by the letter the user guessed.
2. When the game is over according to **`processUserGuess`**, it determines if the user won or not, prints out the corresponding text:
    a. If the user won, it prints out "you guessed the word: " + secretWord.
    b. If the user lost, it prints out **`"you're hung!!"`** and then on a new line **`"word is " + secretWord`**.
3. After printing out whether the user won, it also prints "**`you made GUESSES guesses with MISSES misses`**" where GUESSES is the number of guesses by the user and MISSES is the number of misses by the user.
    a. Your code can keep track of the number of misses as the game is played, or it can calculate the number of misses at the end by subtracting the number of misses left from the total number of misses allowed.

Returns:
Type: bool
Returns True if the user won the game and False otherwise.

Main Program Block

The main program block of Hangman.py should ***use a while loop*** to implement the following functionality when Hangman.py is run:

- Running Hangman.py should begin a “session” in which the user will play one or more games of Hangman.

    - The user should always play at least one game of Hangman.
- After each game, the user should be prompted if they want to play again.
- Once the user indicates that they are ready to stop playing, win-loss summary stats for the "session" should be printed.
    - Use the return value of **`runGame`**, which indicates a win or loss in a game, to accumulate a win-loss count for the "session".

See the Hangman runs below for examples.

Example Hangman Runs

Session with 2 Games

In the first game, the user successfully guesses the word "carpeting" on hard mode. In the second game, the user fails to guess the word "leigh" on easy mode.

```
How many misses do you want? Hard has 8 and Easy has 12
(h)ard or (e)asy> h
you have 8 misses to guess word

letters you've guessed:
misses remaining = 8
_ _ _ _ _ _ _ _ _
letter> a

letters you've guessed: a
misses remaining = 8
_ a _ _ _ _ _ _ _
letter> e

letters you've guessed: a e
misses remaining = 8
_ a _ _ e _ _ _ _
letter> o
you missed: o not in word

letters you've guessed: a e o
misses remaining = 7
_ a _ _ e _ _ _ _
letter> u
you missed: u not in word
```

```
letters you've guessed: a e o u
misses remaining = 6
_ a _ _ e _ _ _ _
letter> i

letters you've guessed: a e i o u
misses remaining = 6
_ a _ _ e _ i _ _
letter> t

letters you've guessed: a e i o t u
misses remaining = 6
_ a _ _ e t i _ _
letter> r

letters you've guessed: a e i o r t u
misses remaining = 6
_ a r _ e t i _ _
letter> n

letters you've guessed: a e i n o r t u
misses remaining = 6
_ a r _ e t i n _
letter> g

letters you've guessed: a e g i n o r t u
misses remaining = 6
_ a r _ e t i n g
letter> p

letters you've guessed: a e g i n o p r t u
misses remaining = 6
_ a r p e t i n g
letter> c
you guessed the word: carpeting
you made 11 guesses with 2 misses

Do you want to play again? y or n> y
How many misses do you want? Hard has 8 and Easy has 12
(h)ard or (e)asy> e
you have 12 misses to guess word

letters you've guessed:
```

```
misses remaining = 12

_ _ _ _ _
letter> a
you missed: a not in word

letters you've guessed: a
misses remaining = 11

_ _ _ _ _
letter> o
you missed: o not in word

letters you've guessed: a o
misses remaining = 10

_ _ _ _ _
letter> u
you missed: u not in word

letters you've guessed: a o u
misses remaining = 9

_ _ _ _ _
letter> i

letters you've guessed: a i o u
misses remaining = 9
_ _ i _ _
letter> e

letters you've guessed: a e i o u
misses remaining = 9
_ e i _ _
letter> r
you missed: r not in word

letters you've guessed: a e i o r u
misses remaining = 8
_ e i _ _
letter> n
you missed: n not in word

letters you've guessed: a e i n o r u
misses remaining = 7
_ e i _ _
letter> s
```

```
you missed: s not in word

letters you've guessed: a e i n o r s u
misses remaining = 6
_ e i _ _
letter> t
you missed: t not in word

letters you've guessed: a e i n o r s t u
misses remaining = 5
_ e i _ _
letter> p
you missed: p not in word

letters you've guessed: a e i n o p r s t u
misses remaining = 4
_ e i _ _
letter> h

letters you've guessed: a e h i n o p r s t u
misses remaining = 4
_ e i _ h
letter> c
you missed: c not in word

letters you've guessed: a c e h i n o p r s t u
misses remaining = 3
_ e i _ h
letter> t
you already guessed that

letters you've guessed: a c e h i n o p r s t u
misses remaining = 3
_ e i _ h
letter> k
you missed: k not in word

letters you've guessed: a c e h i k n o p r s t u
misses remaining = 2
_ e i _ h
letter> b
you missed: b not in word
```

```
letters you've guessed: a b c e h i k n o p r s t u
misses remaining = 1
_ e i _ h
letter> g

letters you've guessed: a b c e g h i k n o p r s t u
misses remaining = 1
_ e i g h
letter> w
you're hung!!
word is leigh
you made 16 guesses with 12 misses

Do you want to play again? y or n> n
You won 1 game(s) and lost 1
```

Session with 1 Game

In the only game played, the user fails to guess the word “skyhook” on hard mode.

```
How many misses do you want? Hard has 8 and Easy has 12
(h)ard or (e)asy> h
you have 8 misses to guess word

letters you've guessed:
misses remaining = 8
_ _ _ _ _ _ _
letter> a
you missed: a not in word

letters you've guessed: a
misses remaining = 7
_ _ _ _ _ _ _
letter> e
you missed: e not in word

letters you've guessed: a e
misses remaining = 6
_ _ _ _ _ _ _
letter> o

letters you've guessed: a e o
```

```
misses remaining = 6
_ _ _ _ o o _
letter> i
you missed: i not in word

letters you've guessed: a e i o
misses remaining = 5
_ _ _ _ o o _
letter> u
you missed: u not in word

letters you've guessed: a e i o u
misses remaining = 4
_ _ _ _ o o _
letter> a
you already guessed that

letters you've guessed: a e i o u
misses remaining = 4
_ _ _ _ o o _
letter> s

letters you've guessed: a e i o s u
misses remaining = 4
s _ _ _ o o _
letter> t
you missed: t not in word

letters you've guessed: a e i o s t u
misses remaining = 3
s _ _ _ o o _
letter> r
you missed: r not in word

letters you've guessed: a e i o r s t u
misses remaining = 2
s _ _ _ o o _
letter> n
you missed: n not in word

letters you've guessed: a e i n o r s t u
misses remaining = 1
s _ _ _ o o _
```

```
letter> d
you're hung!!
word is skyhook
you made 10 guesses with 8 misses

Do you want to play again? y or n> n
You won 0 game(s) and lost 1
```

Testing on your own: CheckMyFunctions.py

The starter code has a file called CheckMyFunctions.py. Since the autograder relies on runGame() to start the game and test the code and the hand grading requires running the main program block, it does not make sense for you to also test the code in the main block. Therefore, we have provided the CheckMyFunctions.py file with some starting code to help you with this.

We will not look at this file nor grade it; it is only there to help you with testing the code. You may use it (or not) as you see fit.

Submitting and Grading

1. Log in to Gradescope.
2. Under the CompSci 101 dashboard, click Assignment 4: Hangman.
3. Click Submit Programming Assignment, click “Click to browse”, and select the required files. Click “Browse Files” again to select and submit more files.
4. Once all files are selected, click Upload.

You can submit more than once. Your last submission will be graded.

This assignment is out of 50 points.

Autograded portion [38 points]

- [4] handleUserInputDifficulty
    - Gives the user the option for easy or hard mode
    - Returns the correct value
- [6] getWord
    - secretWord is chosen randomly from words
    - secretWord is the correct length
    - Returns the secretWord
- [6] createDisplayString
    - lettersGuessed are in sorted order

    - Returns a string contains the lettersGuessed, hangmanWord, missesLeft
    - Returned string is in the correct format
- [8] handleUserInputLetterGuess
    - Prints out the value in the displayString variable
    - Takes in user input to guess a letter
    - Returns the user's guessed letter
    - Checks if the user entered a repeated letter and prompts the user to enter a new letter until a non-repeated letter is entered
- [5] updateHangmanWord
    - Updates hangmanWord with guessedLetter based on the secretWord
    - Returns the updated hangmanWord
- [5] processUserGuess
    - Updates the value of missesLeft
    - Updates whether or not the user guessed wrong
    - Returns the correct values in a list
- [4] runGame
    - Calls the other functions to run the game
    - Takes in a single parameter with the filename

Hand-graded portion [12 points]

- [3] runGame
    - [2] A random word length was chosen between 5 and 10 (inclusive)
    - [1] Returns True if the user won the game, False otherwise
- [5] Hand-Running Hangman.py: Single Game
    - [1] Game begins without errors
    - [1] Game works mostly without errors or incorrect things when the user wins
    - [1] Game works mostly without errors or incorrect things when the user loses
    - [1] Prints correct single-game summary statistics
    - [1] Game consistently works properly
- [2] Hand-Running Hangman.py: Multiple-Game Session
    - [1] Prompts user if they want to play again and follows the user's request
    - [1] Prints correct multi-game summary statistics
- [2] Rest of submission
    - [2] points for name at the top and docstring comments (including for any optional helper functions)

Note About While Loops

In the instructions above, it is recommended that you use while loops. However, it is possible that a bug in your code may cause an "infinite loop", in which case the while loop will never be exited and the program will run forever. When this occurs, you can use the red square button next to the console in Pycharm in order to terminate the program immediately: