Problem Set 2

Out: 9/10/18 5:00pm

Due: 9/17/18 5:00pm

This problem set explores bisection search, as well as the use of abstraction to reduce a complex problem to manageable (and reusable) pieces, and provides an opportunity to practice working with files and strings.

Collaboration

You may collaborate on this assignment, but must note who you collaborated with at the top of your submitted file (it is never okay to provide a complete problem set solution to another student).

Header format for submitted assignments

All submitted assignments should have a header which contains, at a minimum, your name and a list of everyone you collaborated with on the assignment. For example, it might look like this:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Eric Hoppmann
@collaborators: Testudo
"""

<code>
```

Submission of completed problem sets

Your submission will be the last commit which was pushed to GitHub prior to the submission deadline.

If you are electing to use one of your 2 late submissions for this problem set, you must notify me prior to the expiration of the 2 day late period (for example, if the original deadline was 9/17 at 5pm, you must notify me by email at hoppmann@umd.edu prior to 9/19 at midnight that you have elected to use a late day, in which case your final commit which is pushed to GitHub prior to 9/19 at 5pm will be the submission which is graded).

Part 1: Bisection search solution to 'pay off in one year' (3

pts)

In the lab, we limited the monthly payments to multiples of 10. This is for performance reasons - for large balances / interest rates, it will begin to run very slowly if you change the increment for the monthly payments to 0.01 (try it!). The goal here is to use bisection search to efficiently locate the correct answer to within a tolerance of one cent.

The bisection method is a root finding method that searches some range, bounded by `a` and `b`, by repeatedly bisecting it, until it converges on a solution within the specified tolerance `epsilon`.

In our case, the initial lower bound, `a` might be given by 0 dollars, while the initial upper bound `b` might be given by the total loan amount (lump sump payoff), and epsilon will be 0.01 (one cent).

Note: If you do not use bisection search, your program will take too long to run, and you will not receive credit.

Here is an example test case to check your code:

```
Enter the starting balance: 500000
Enter the APR: 18
Monthly Payment:  45523.18900823593
```

Part 2: Hangman (7 pts)

Getting Started

Before beginning, ensure that you are able to run the `ps2_b.py` file and get the following output:

```
Loading word list from file: words.txt
8812 words loaded
```

The Game

The goal of this problem set is to implement a function called `hangman` that will allow the user to play a game of hangman against the computer. The computer picks the word, and the user tries to guess letters which are in the word.

Here's the description of the final result. We will break this down into manageable pieces…**don't be intimidated and keep reading**

1. The computer will select a secret word at random from the list of available words in `words.txt` (words in `words.txt` are all lowercase)

2. The user is given a number of guesses equal to the length of the secret word + 4
3. The interactive game begins. The user enters a guess and the computer either:
    - Reveals the letter if it exists in the secret word
    - Reduces the user's remaining guesses by 1
4. The game ends when the user has guessed the secret word, or the user is out of guesses.

Part 1: Three helper functions

In the first part of the problem, we are going to create three helper functions that will help drive the functionality of the `hangman` function. By splitting these subtasks out of the main function, we make the problem more manageable and simplify debugging. This is a common approach in computational problem solving!

Determine whether the word has been guessed

In this part, the goal is to write the `is_word_guessed` function. The function takes two parameters, a string `secret_word` and a string containing the letters which have been guessed `letters_guessed` (e.g. `'abc'` if the user has guessed a, b, and c. You can assume that all letters in `letters_guessed` are valid lowercase letters). It returns a bool, True if all letters in the word have been guessed, otherwise False.

Hint: Use the Python `in` method which checks for item membership. You can use the `not` operator to toggle a boolean (i.e. from True to False or vice versa). For example, the following code can be used to check if a character is (or isn't) part of a string:

```
is_a_member = 'a' in 'apple'
print(is_a_member)  # --> True
isnt_a_member = 'a' not in 'apple'
print(isnt_a_member)  # --> False
```

Example test case:

```
secret_word = 'apple'
letters_guessed = 'aep'
is_word_guessed(secret_word, letters_guessed) # --> False
letters_guessed += 'l'
is_word_guessed(secret_word, letters_guessed) # --> True
```

Print out the secret word showing letters that have been guessed

In this part, the goal is to write the function `print_secret_word_with_letters_guessed`. This function takes two parameters, the string `secret_word` and string `letters_guessed`. It can be assumed that

both are provided in lowercase. The function doesn't return anything…it simply prints out the secret word, with underscores for letters not in `letters_guessed`.

Example test case:

```
secret_word = 'apple'
letters_guessed = 'a'
print_secret_word_with_letters_guessed(secret_word, letters_guessed)
letters_guessed = 'aep'
print_secret_word_with_letters_guessed(secret_word, letters_guessed)
```

Prints:

```
a _ _ _ _
a p p _ e
```

Get all available letters

In this part, the goal is to write the function `get_available_letters`. This function takes one parameter, the string of letters which have been guessed `letters_guessed` (it can be assumed that these are all lowercase), and returns a string of letters which the user has not yet guessed.

- Hint 1: You may find the `string.ascii_lowercase` method useful.

Example test case:

```
available_letters = get_available_letters('abdz')
print(available_letters)
```

Which prints:
`cefghijklmnopqrstuvwxy`

Part 2: The game

Now that the helper functions have been built, you can turn to implementing the game `hangman`, which accepts only one parameter: `secret_word`. Initially you can call `hangman` with secret words that you manually set to make debugging easier - later, you can call it using the provided word list (the code skeleton provided will do this automatically if you run the entire python file, that's what the `if __name__ == '__main__':` block of code is for).

Calling `hangman(secret_word)` starts an interactive game of hangman. Make sure to use the three helper functions that you wrote in the previous part to implement `hangman`!

Game architecture

1. The computer randomly selects a word from the word list
2. The user starts with a number of guesses equal to the length of the word + 4
3. Each round, the user has the opportunity to guess a letter. The user is immediately told whether or not the letter was in the word, and their progress in the game is printed by calling the `print_secret_word_with_letters_guessed` function.
4. A line of dashes should be printed between each round to help visually separate the rounds.

User input

1. The user inputs one character per round. If the user enters anything other than a lower or uppercase letter, (either a non-letter character, or multiple characters, or no characters), ask the user to enter another guess without docking them a guess.
2. The entered character should be converted to lowercase before further processing.

User input requirements

1. You should ensure that the user only inputs one character. If the user inputs multiple characters, or none, or any character which is not a letter (including for example a symbol or a number), you should tell the user that their guess is invalid (but not reduce their number of remaining guesses). However, capital letters *should* be accepted (and converted to lowercase by your program).

- Hint: you may find the `'some_string'.lower()` method useful!

Game end

- The game ends when the user has either correctly guessed the word, or runs out of guesses.
- The correct word should be printed when the game ends, along with a message either telling the user they lost or congratulating them for winning.

Example winning game

```
The secret word is 5 letters long
You have 9 guesses to guess the secret word

--------------------
Available letters:  abcdefghijklmnopqrstuvwxyz
Guessed letters:

_ _ _ _ _
Please guess a letter: a
You got one!
--------------------
Available letters:  bcdefghijklmnopqrstuvwxyz
Guessed letters:
a _ _ _ _
Please guess a letter: e
You got one!
--------------------
Available letters:  bcdfghijklmnopqrstuvwxyz
Guessed letters:
a _ _ _ e
Please guess a letter: i
Sorry, that letter is not in the secret word
You have 8 guesses remaining
--------------------
Available letters:  bcdfghjklmnopqrstuvwxyz
Guessed letters:
a _ _ _ e
Please guess a letter: l
You got one!
--------------------
Available letters:  bcdfghjkmnopqrstuvwxyz
Guessed letters:
a _ _ l e
Please guess a letter: p
You got one!
Congratulations, you've won! The secret word was:  apple
```

Example losing game

```
The secret word is 5 letters long
You have 9 guesses to guess the secret word

--------------------
Available letters:  abcdefghijklmnopqrstuvwxyz
Guessed letters:

_ _ _ _ _
```

```
Please guess a letter: a
You got one!
--------------------
Available letters:  bcdefghijklmnopqrstuvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: e
Sorry, that letter is not in the secret word
You have 8 guesses remaining
--------------------
Available letters:  bcdfghijklmnopqrstuvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: i
Sorry, that letter is not in the secret word
You have 7 guesses remaining
--------------------
Available letters:  bcdfghjklmnopqrstuvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: p
Sorry, that letter is not in the secret word
You have 6 guesses remaining
--------------------
Available letters:  bcdfghjklmnoqrstuvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: u
Sorry, that letter is not in the secret word
You have 5 guesses remaining
--------------------
Available letters:  bcdfghjklmnoqrstvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: p
Already guessed!
--------------------
Available letters:  bcdfghjklmnoqrstvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: l
Sorry, that letter is not in the secret word
You have 4 guesses remaining
--------------------
Available letters:  bcdfghjkmnoqrstvwxyz
Guessed letters:
```

```
_ a _ _ _
Please guess a letter: f
Sorry, that letter is not in the secret word
You have 3 guesses remaining
--------------------
Available letters:  bcdghjkmnoqrstvwxyz
Guessed letters:
_ a _ _ _
Please guess a letter: r
You got one!
--------------------
Available letters:  bcdghjkmnoqstvwxyz
Guessed letters:
_ a r r _
Please guess a letter: m
Sorry, that letter is not in the secret word
You have 2 guesses remaining
--------------------
Available letters:  bcdghjknoqstvwxyz
Guessed letters:
_ a r r _
Please guess a letter: b
Sorry, that letter is not in the secret word
You have 1 guesses remaining
--------------------
Available letters:  cdghjknoqstvwxyz
Guessed letters:
_ a r r _
Please guess a letter: c
Sorry, that letter is not in the secret word
You have 0 guesses remaining
Sorry, you've lost. The secret word was:  harry
```