Contents

1 Basic Test Results

```
Starting tests...
Tue Nov 17 22:08:17 IST 2015
c98303b6107c222b9e7d373cc51b753e205be48b  -


Archive:  /tmp/bodek.rfUboC/intro2cs/ex4/elinorperl/presubmission/submission
  inflating: src/hangman.py
  inflating: src/README

Testing README...
Done testing README...

Running presubmit tests...
6 passed tests out of 6
result_code    ex4    6    1
Done running presubmit tests

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.
```

2 README

```
elinorperl
329577464
Elinor Perl

I discussed the exercise with: Yoav Gross, Nophar Sarel, and lab support.

============================
=  README for ex4: Hangman  =
============================


==================
=  Description:  =
==================
In this project, we created hangman. The user was unable to enter undefined characters, re-enter letters he already used. Th

======================
=  Special Comments  =
======================

I used stackoverflow.com and http://www.secnetix.de/olli/Python
```

3 hangman.py

```
import hangman_helper

MAX_ERRORS = hangman_helper.MAX_ERRORS
WIN_MSG = hangman_helper.WIN_MSG
LOSS_MSG = 'You have run out of guesses, the word was: '
ALREADY_CHOSEN_MSG = 'You have already chosen '
NON_VALID_MSG = 'Please enter a valid letter'
HINT_MSG = 'Consider choosing: '
DEFAULT_MSG = ''
HINT = 1
LETTER = 2
PLAY_AGAIN = 3
UNDERSCORE = "_"

def update_word_pattern (word ,pattern ,letter):
    """
    :param word: I input a word to use as the base of the game for which
    the user must guess letters to make up the word.
    :param pattern: The pattern is essentially the shell for the letters
    guessed, inserting the letters that are found in our word and disregarding
    the letters that don't apply.
    :param letter: The letter is the letter that is guessed toward the word.
    """
    pattern_list = list(pattern)
    list(word)
    for index, test_letter in enumerate(word):
        if word[index] == letter:
            pattern_list[index] = test_letter
    return ''.join(pattern_list)


def filter_words_list(words, pattern, wrong_guess_lst):
    """
    :param words: This parameter is the list of words hangman takes for its
    game.
    :param pattern: This is the current pattern which starts off as the the
    length of the word hangman took for it's game, and as the user guesses the
    right letters, it updates taking the shape of the word.
    :param wrong_guess_lst: This is the list of all the guesses from the user
    that didn't fit into the word that hangman defined.
    In this function, I operated on the "negative". I assumed that all words
    could be added to the list, and represented the reasons to "deny" their
    entrance to the list according to the conditions that were set.
    """
    potential_words = []
    for word in words:
        add_to_list = True
        if len(word) != len(pattern):
            add_to_list = False
        else:
            for letter in wrong_guess_lst:
                if letter in word:
                    add_to_list = False
            for index in range(len(pattern)):
                if pattern[index] != word[index] and pattern[index] != \
                        UNDERSCORE:
                    add_to_list = False
                elif pattern[index] == word[index]:
                    for index2 in range(len(word)):
```

```
                        if word[index2] == pattern[index] and pattern[index] \
                                                != pattern[index2]:
                            add_to_list = False
        if add_to_list:
            potential_words.append(word)
    return potential_words


def choose_letter(words,pattern):
    """
     This function takes a list and combines the elements into one string.
     Using dictionary and loop, it will count the amount of times each letter
      appears only if the letter does not appear in the pattern prior. I
       then defined a function to find the maximum value of the keys
      and return the key with the highest value.
    """
    joined_list = ''.join(words)
    counter = {}
    for i in joined_list:
        if i not in pattern:
            if not counter.get(i):
                counter[i] = 1
            else:
                counter[i] += 1
    value_list=list(counter.values())
    key_list=list(counter.keys())
    max_key =  key_list[value_list.index(max(value_list))]
    return max_key

def run_single_game(word_list):
    """
    In this function, I defined the list of words, taking it from function
    "hangman_helper", created a list adding the wrong guesses of the
    user into the list I assigned to it and defined the pattern to be the
    size of the random word. I defined the conditions for the input of the
    letter and defined the following actions the program would take.
    """
    word = hangman_helper.get_random_word(word_list)
    wrong_guesses = []
    error_count = 0
    pattern = UNDERSCORE*len(word)
    user_message = hangman_helper.DEFAULT_MSG
    while len(wrong_guesses) < hangman_helper.MAX_ERRORS and word != pattern:
        hangman_helper.display_state(pattern,error_count,wrong_guesses,\
                                     user_message)
        input_type, user_input = hangman_helper.get_input()
        if input_type == hangman_helper.HINT:
            filtered_list = filter_words_list(word_list,pattern,wrong_guesses)
            letter_hint = choose_letter(filtered_list,pattern)
            user_message = hangman_helper.HINT_MSG + letter_hint
            continue
        elif len(user_input) > 1 or not(user_input.islower()):
            user_message = hangman_helper.NON_VALID_MSG
            continue
        elif user_input in wrong_guesses:
            user_message = hangman_helper.ALREADY_CHOSEN_MSG + user_input
            continue
        elif user_input in pattern:
            user_message = hangman_helper.ALREADY_CHOSEN_MSG + user_input
            continue
        elif user_input in word:
            pattern = update_word_pattern(word,pattern,user_input)
            user_message = hangman_helper.DEFAULT_MSG
            continue
        else:
            wrong_guesses.append(user_input)
            error_count += 1
            user_message = hangman_helper.DEFAULT_MSG
```

```
            continue
        if word == pattern:
            user_message = hangman_helper.WIN_MSG
        else:
            user_message = hangman_helper.LOSS_MSG + word
        hangman_helper.display_state(pattern,error_count,wrong_guesses,\
                                     user_message,True)


def main():
    """
    Main is the function from which the whole graphic side operates. I called
    the list of words to use for hangman, and called run_single_game and if
    input was PLAY AGAIN and true, to restart the game and allow input.
    """
    list_of_words = hangman_helper.load_words()
    run_single_game(list_of_words)
    input_type, user_input = hangman_helper.get_input()
    while input_type == hangman_helper.PLAY_AGAIN and user_input:
        run_single_game(list_of_words)
        input_type, user_input = hangman_helper.get_input()

if __name__ == "__main__":
    hangman_helper.start_gui_and_call_main(main)
    hangman_helper.close_gui()
```