|  |  |  |
| --- | --- | --- |
|  | |  |
|  | |  |
| 2810ICT – Software Technologies | |  |
| Assignment 1 Documentation | |  |
| Submission Date: 21 August 2017 | |  |
| Student Name: Luke Reynolds | Student Name: Jordan Schurmann |  |
| Student Number: 507xxxx | Student Number: 5073958 |  |
| Contribution: 50% | Contribution: 50% |  |
|  | |  |

|  |
| --- |
|  |

**Contents**

[1.0 Problem Statement 3](#_Toc490154679)

[1.1 Initial Problem Analysis 3](#_Toc490154680)

[2.0 User Requirements 4](#_Toc490154681)

[3.0 Software Requirements 5](#_Toc490154682)

[4.0 Software Design 6](#_Toc490154683)

[4.1 High Level Design – Logical Block Diagram 6](#_Toc490154684)

[4.2 List of Functions 7](#_Toc490154685)

[4.3 List of Data Structures 10](#_Toc490154686)

[4.4 Detailed Design 11](#_Toc490154687)

[4.5 Configuration Management and Version Control 13](#_Toc490154688)

[5.0 Unit tests 15](#_Toc490154689)

[5.1 Unit Test Execution 20](#_Toc490154690)

[6.0 Requirements Acceptance Tests 22](#_Toc490154691)

[7.0 User Instructions 24](#_Toc490154692)

[7.1 Execution example 24](#_Toc490154693)

# Problem Statement

The goal of a ladder-gram software program is to transform an input source word into the input target word in the least number of steps. During each step, the software replaces one letter in the previous word so that a new word is formed, but without changing the positions of the other letters. All words in the steps exist in the supplied dictionary (dictionary.txt). For example, we can achieve the alchemist's dream of changing “lead” to “gold” in 3 steps (lead->load->goad->gold), or “hide” to “seek” in 6 steps (hide->side->site->sits->sets->sees->seek).

The software code is not optimised or documented.

## Initial Problem Analysis

The key problem with the current Word Ladder implementation, detected through debugging, is that the program is not taking words with the greatest number of letter matches to the target word as priority. Each pattern match starts with the next word in sequence whether this has the largest number of letter matches or not. It then attempts to find a path from there and is not optimal. This sometimes results in the recursion limit being reached.

The solution is to focus the code to use words with the greatest number of letter matches to the target word as priority. This required the current software implementation to be updated to include modified/additional steps to (i) sort words such that the words with the highest number of matching letters appear in the list first, (ii) only pattern match on the letters that do not exist in a list of letters matched, so that we get the next words in sequence and not all words that are one off each letter in the new start word, and (iii) for each new iteration using the word with the highest number of letter matches, update the list of letters matched with the new position of letter matches. This way we progressively build up the number of match letters and arrive at the target word with less iterations. For example:

* Iteration 1 – lettermatched = []
  + Build subsequent pattern for all word positions
* Iteration 2 – lettermatched = [0] – First letter matches target
  + Build subsequent pattern excluding position 1 from word pattern
* Iteration 3 – lettermatched = [0,5] – First and sixth letter matches target
  + Build subsequent pattern excluding position positions 1 and 6 from word pattern
* etc

# User Requirements

The following outlines the user requirements for the program:

1. At the start of the game, user must provide a data file containing a dictionary of the words for the game. This file name is user entered and includes the file extension.
2. At the start of the game, the user has the option of providing a data file containing a list of words that are to be excluded from the dictionary. This file name is user entered and includes the file extension.
3. At the start of the game, the user shall be able to enter a start word for the game.
4. At the start of the game, the user shall be able to enter a target word for the game.
5. At the start of the game, the user will have the option of selecting the shortest path to the target word. The default is the longest path
6. The start and target words must exist within the input dictionary and take into consideration any excluded words. The start word can not be excluded.
7. All input will be validated.
8. The system shall correctly calculate the path to the target word and display the number of steps taken, plus the words used in the path, at the end of the game.
9. All files will be validated for existence and whether they are empty. Only words that are alphabetic will be read into the list of words.
10. The start and target words must be the same length, and alphabetic.
11. The indicator to select to include a list of words to exclude must be “Y” or “N”.
12. The indicator to select the shortest path must be “Y” or “N”

# Software Requirements

The following outlines the software requirements for the program:

1. The program shall read in a single “not empty” file input by the user. The program will reprompt until a candidate file is found. This file must have a valid file name and not be empty. The file must be a text file
2. The program shall optionally prompt the user to supply a list of excluded words. If the decision is made to provide a list of excluded words then the program will reprompt until a candidate file is found. This file must have a valid file name and not be empty.
3. The program shall prompt the user to enter a starting word. This word must exist in the input file. The word must be alphabetic and longer than one character.
4. The program shall prompt the user to enter a target word. This word must exist in the input file and not be an excluded word. The word must be alphabetic and the same size as the start word.
5. The program shall prompt the user as to whether they want the system to calculate the shortest path. The default value is the longest path.
6. The program shall display the word path according to the parameters input, together with the size of the path.
7. The program will read in the list of words and exclude any words that have been provided in the excluded word list.
8. The program shall find the path from one word to another word by changing a letter at a time. For example, it will find the path of words from hide->side->site->sits->sets->sees->seek. All words used within the word ladder must be contained in the list of words for the game.
9. If a path from the start word to the target word can not be found then display “No Path Found”.

# Software Design

## High Level Design – Logical Block Diagram

High level diagram of the program logic is shown below:



## List of Functions

The following table lists all the functions used in the game:

| **Module / Class** | **Function** | **Parameters / Data Types** | **Side effects** | **Description** |
| --- | --- | --- | --- | --- |
| Word\_Ladder | same | :param item: Word being compared to the Target letter by letter  :type item: str  :param target: Target Word  :type target: str  :return: Integer of the number of letter matches for the word  :rtype: int | Not applicable | Function returns the number of letters for each item that match that target word. |
|  | build | :param pattern: Pattern changes on each evocation - e.g. lead: ".ead", "l.ad", "le.d", "lea.")  :type pattern: str  :param words: List of all words  :type words: list  :param seen: Dictionary with words and a value of true if they have been identified  :type seen: dict  :param list: List of words that match the current pattern and are not in seen  :type list: list  :return: Words matching pattern  :rtype: list | Not applicable | Function builds a list of word that match the pattern provided as input. |
|  | find | :param start: Current starting word - changes as this is recursively called  :type start: str  :param words: List of words  :type words: List  :param seen: List of words current processed - seen  :type seen: dict  :param target: Target word  :type target: str  :param path: Current path to the target word  :type path: List  :return: True if found or False if no path for current item  :rtype: Boolean | Not applicable | Function recursively iterates over the each word, gradually moving closer to the target. |
|  | valid\_file | :param fname: File name input  :type fname: str  :return: String indicating whether file data is valid  :rtype: str | Not applicable | Function to validate input files. |
|  | make\_word\_list | :param start: Start word input  :type fname: str  :param lines: list of normal dictionary words  :type lines: list  :param excluded: list of excluded dictionary words  :type excluded: list  :return: list of words for game  :rtype: list | Not applicable | Function to build a word file catering for any excluded words. |
|  | valid\_start | :param start: Start word input  :type start: str  :param lines: List of dictionary words  :type lines: list  :return: String indicating whether data is valid  :rtype: str | Not applicable | Function to validate input for start word. |
|  | valid\_target | :param start: Start word input  :type start: str  :param target: Target word input  :type target: str  :param words: List of dictionary words  :type words: list  :return: String indicating whether data is valid  :rtype: str | Not applicable | Function to validate input for target word. |
|  | valid\_flag | :param flag: String indicating yes or no  :type flag: str  :return: String indicating whether data is valid  :rtype: str | Not applicable | Function to validate y/n options. |

## List of Data Structures

List of all data structures in the software. (eg linked lists, trees, arrays etc)

| **Name** | **Structure Type** | **Description** | **Members** | **Used by** |
| --- | --- | --- | --- | --- |
| matchedletters | List | List containing the numeric position of matched letters when comparing the starting iteration value to the target word. | List of integers | find() |
| list | List | List of words matching a particular pattern and how many letters the word matches | List of pairs of words and a count of the letters the word matches in the target | find()  build()  main program |
| seen | Dict | Dictionary containing word and whether it has been seen | Dictionary of words and whether the word has been seen (true value otherwise null) | find()  build()  main program |
| path | List | List containing words in the path to the target word | List of words | find()  main program |
| excluded | List | List of excluded words | List of excluded words | make\_word\_list()  main program |
| words | List | List of words for the game | List of words | find()  main program |

## Detailed Design

Pseudocode for all non-standard and non-trivial algorithms that operate on datastructures are defined below:

| **Function Name** | **Description** | **Psuedocode** |
| --- | --- | --- |
| same() | Returns the number of letters that match between two words. | For each letter in item word and target word  Compare each letter at each index for both words  If letters are the same at the same index  Increment counter of letters matched  Return the count of letters matched |
| build() | Reads the game list of words and builds a new list of a list of words for this iteration, that match the word pattern and have not already been processed.  Each word is returned and appended to the list in the find() function. | For each word in the list of game words  If the word matches the letter pattern  If the word is not is in the list as having been previously processed “seen”  If the word is not already in the list  Return the word |
| find() | This function is the main processing function that recursively searches for the best path | Read input parameters  If the Start and Target Words have more than zero letter matches  Create array of the indexes of each matched letter  For each index in the Starting word  If the index is not in the array of matched letters  Create a pattern and call the Build() function  Add the found word to a list of words to review  If the list returned is empty (No words found for match)  Quit out of the recursive loop (no path found)  Create a sorted list calling the Same() function for every word in the list  (This list will have the word and the number of matching letters in that word. The list is sorted on the number of matching letters.) Specify reverse to sort the list by the highest number of letter matches first.  For each item and letter count in the list  If letter count is within one of the length of the target word  Append the word to the path  Exit  Mark the word as having been seen  For each item and letter count in the list  Add the item to the path  Call find() recursively  If path found  exit  else  remove the item from the path. |

## Configuration Management and Version Control

For this project the team utilised local GIT version control repositories to manage local code and a shared project repository on GITHUB to shared code between the team members. The local code repositories are integrated into the PyCharm IDE’s of both team members.

Team members would take local copies of the Master code on GITHUB, via a local Pull command, and work on the updated code. Changes would be made locally and commited to the Local GIT repository. When the changes were complete this would be committed and Pushed to the Master repository on GITHUB.

A Push to the GITHUB would not be successful if there were changes already on the GITHUB repository. The team member would have to first Pull / Merge the changes into the local copy, correct any code discrepancies and then complete the code push to GITHUB.

This process is illustrated in the following diagram.



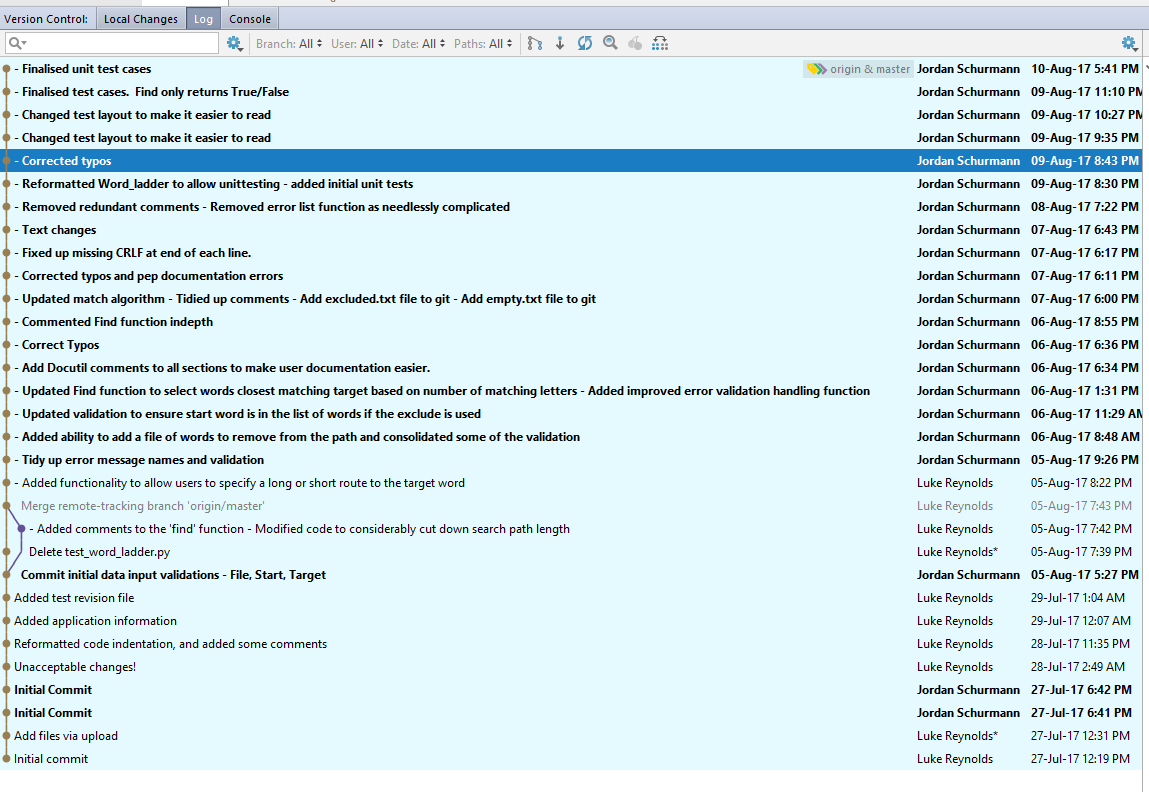
The project repository is a public repository and the link for this repository is:

<https://github.com/lreynolds188/LadderGram#laddergram-python>

Ideally this repository would be a private repository to control who could see the repository contents, however for this project this is not an issue as any copying could be detected via the date of the commits made for the particular piece of source.

Code was shared and pushed / pulled from this repository.

The following screen shot illustrates the commit log for the version control project.



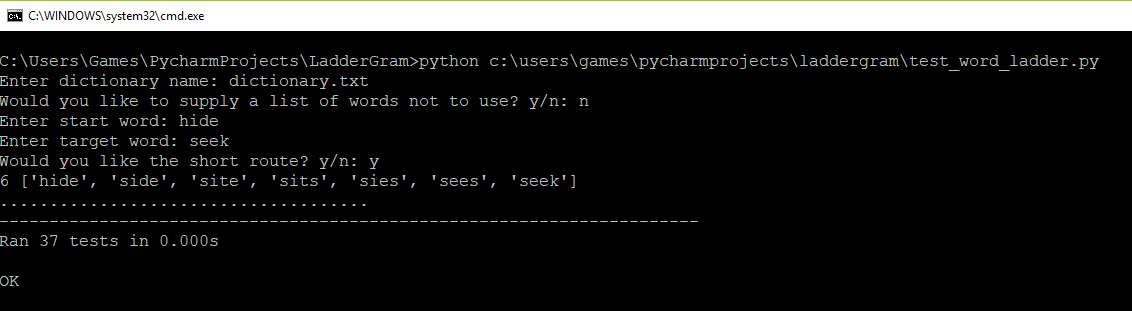
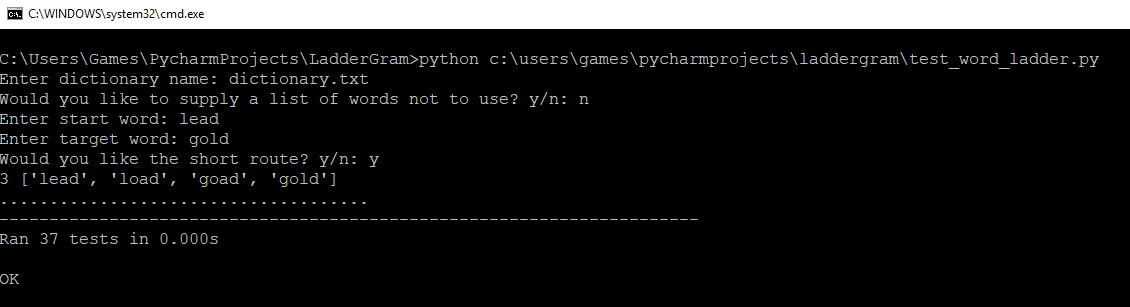
# Unit tests

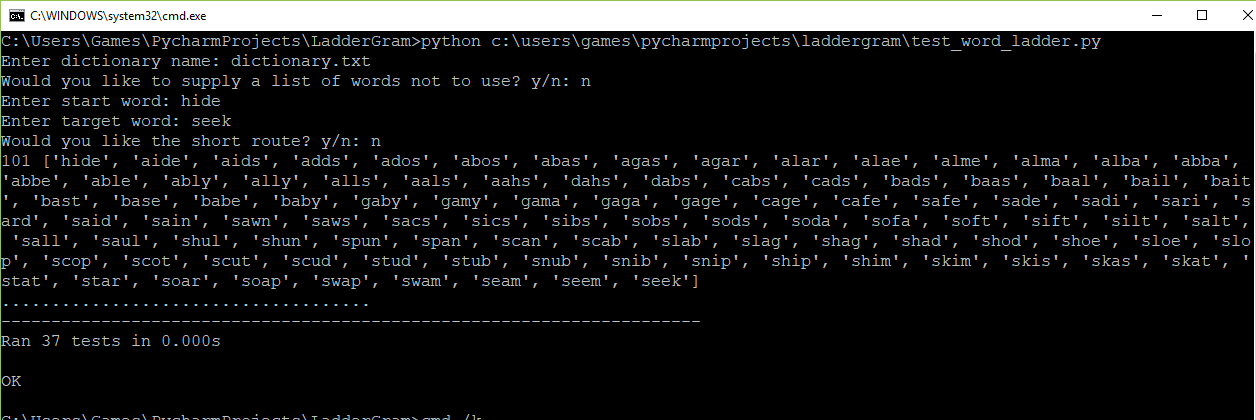
The following unit tests were developed and executed for the program:

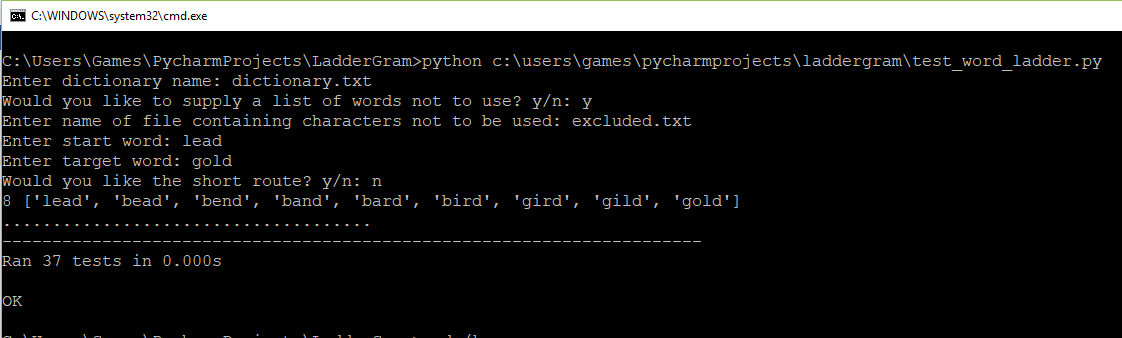
| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| **1.0** | **Same Function** | | |
| 1.1 | # Test no matching letters  item = 'hide'  target = 'seek' | 0 returned | 0 returned |
| 1.2 | # Test 1 matching letters  item = 'hide'  target = 'sits' | 1 returned | 1 returned |
| 1.3 | # Test 2 matching letters  item = 'hide'  target = 'hits' | 2 returned | 2 returned |
| 1.4 | # Test all matching letters  item = 'hide'  target = 'hide' | 4 returned | 4 returned |
| **2.0** | **Build Function** | | |
| 2.1 | # Test 0 matching word - 0 Seen - 0 in list  pattern = '.ode'  words = ['dogs', 'side', 'site', 'tide']  seen = {'ride': True}  list = ['farm', 'lamb'] | [] returned | [] returned |
| 2.2 | # Test 2 matching words - 0 Seen - 0 in list  pattern = '.ide'  words = ['dogs', 'side', 'site', 'tide']  seen = {'ride': True}  list = ['farm', 'lamb'] | ['side', 'tide'] returned | ['side', 'tide'] returned |
| 2.3 | # Test 2 matching words - 1 Seen - 0 in list  pattern = '.ide'  words = ['dogs', 'side', 'site', 'tide']  seen = {'side': True}  list = ['farm', 'lamb'] | ['tide'] returned | ['tide'] returned |
| 2.4 | # Test 2 matching words - 1 Seen - 1 in list  pattern = '.ide'   words = ['dogs', 'fide', 'ride', 'side', 'tide']  seen = {'side': True}  list = ['farm', 'lamb', 'fide'] | ['ride', 'tide'] returned | ['ride', 'tide'] returned |
| **3.0** | **Find Function** | | |
| 3.1 | # Test path found, Return = True  start = 'lead'  words = ['load', 'goad']  seen = {'lead': True}  target = 'gold'  path = ['lead'] | True Returned | True Returned |
| 3.2 | # Test no path found, Return = False  start = 'lead'  words = ['load', 'goss']  seen = {'lead': True}  target = 'gold'  path = ['lead'] | False Returned | False Returned |
| **4.0** | **Valid\_File Function** | | |
| 4.1 | #Test valid file  fname = 'dictionary.txt' | “0” returned (ok) | “0” returned (ok) |
| 4.2 | # Test file is empty  fname = 'empty.txt' | “Selected file is empty….please reenter” returned | “Selected file is empty….please reenter” returned |
| 4.3 | # Test file name can not be found - invalid file name or doesn't exist  fname = 'no such file' | “Can not find the file….please reenter” returned | “Can not find the file….please reenter” returned |
| 4.4 | # Test enter pressed and no file name entered  fname = '' | “Can not find the file….please reenter” returned | “Can not find the file….please reenter” returned |
| **5.0** | **Valid\_Start Function** | | |
| 5.1 | #Test start word exists in list of words  start = 'hide'  lines = ['hide', 'seek', 'smith'] | “0” returned (ok) | “0” returned (ok) |
| 5.2 | # Test start word not in list of words  start = 'hide'  lines = ['cide', 'seek', 'smith'] | “Start word not in list of words…please reenter” returned | “Start word not in list of words…please reenter” returned |
| 5.3 | # Test start word only has one character  start = 'a'  lines = ['hide', 'seek', 'smith'] | “Start word must contain more than one letter….please reenter” returned | “Start word must contain more than one letter….please reenter” returned |
| 5.4 | # Test start word that is not alphabetic  start = 'h1de'  lines = ['hide', 'seek', 'smith'] | “Start word must only contain letters….please reenter” returned | “Start word must only contain letters….please reenter” returned |
| 5.5 | # Test enter pressed an no start word entered  start = ''  lines = ['hide', 'seek', 'smith'] | “Start word must only contain letters….please reenter” returned | “Start word must only contain letters….please reenter” returned |
| **6.0** | **Valid\_Target Function** | | |
| 6.1 | #Test target word exists in list of words  start = 'hide'  target = 'seek'  words = ['hide', 'seek', 'smith'] | “0” returned (ok) | “0” returned (ok) |
| 6.2 | # Test target word not in list of words  start = 'hide'  target = 'seke'  words = ['hide', 'seek', 'smith'] | “Target word not in list of words….please reenter” returned | “Target word not in list of words….please reenter” returned |
| 6.3 | # Test target word is not the same as the start word  start = 'hide'  target = 'hide'  words = ['hide', 'seek', 'smith'] | “Target word must be different from Start word…please reenter” returned | “Target word must be different from Start word…please reenter” returned |
| 6.4 | # Test target word only has one character  start = 'hide'  target = 's'  words = ['hide', 'seek', 'smith'] | “Target word must be same length as Start word....please reenter" returned | “Target word must be same length as Start word....please reenter" returned |
| 6.5 | # Test target word is not longer than start word  start = 'hide'  target = 'seeks'  words = ['hide', 'seek', 'smith'] | “Target word must be same length as Start word....please reenter" returned | “Target word must be same length as Start word....please reenter" returned |
| 6.6 | # Test target word is not less than start word  start = 'hide'  target = 'sek'  words = ['hide', 'seek', 'smith'] | “Target word must be same length as Start word....please reenter" returned | “Target word must be same length as Start word....please reenter" returned |
| 6.7 | # Test target word that is not alphabetic  start = 'hide'  target = 's3ek'  words = ['hide', 'seek', 'smith'] | “Target word must contain only letters….please reenter” returned | “Target word must contain only letters….please reenter” returned |
| 6.8 | # Test enter pressed an no target word entered  start = 'hide'  target = ''  words = ['hide', 'seek', 'smith'] | “Target word must contain only letters….please reenter” returned | “Target word must contain only letters….please reenter” returned |
| **7.0** | **Valid\_Flag Function** | | |
| 7.1 | # Test valid y flag entered - note case is converted to lower and all front and back spaces are stripped on input  flag = 'y'  self.assertEqual(valid\_yn(flag), 'y') | “Y” returned | “Y” returned |
| 7.2 | # Test valid n flag entered  flag = 'n' | “N” returned | “N” returned |
| 7.3 | # Test alphabetic character other than y or n entered  flag = 'w' | "Please enter letters Y or N only" returned | "Please enter letters Y or N only" returned |
| 7.4 | # Test numeric character entered  flag = '1' | "Please enter letters Y or N only" returned | "Please enter letters Y or N only" returned |
| 7.5 | # Test more than 1 character entered- alphabetic  flag = 'aa' | "Please enter only one character" returned | "Please enter only one character" returned |
| 7.6 | # Test more than 1 character entered- numeric  flag = '123' | "Please enter only one character" returned | "Please enter only one character" returned |
| 7.7 | # Test enter pressed and no flag entered  flag = '' | “Please enter a character” returned | “Please enter a character” returned |
| **8.0** | **Make\_Word\_File Function** | | |
| 8.1 | #Test empty exclusion file  start = 'side'  lines = ['bats', 'cats', 'dogs', 'side']  excluded = [] | ['bats', 'cats', 'dogs', 'side'] returned | ['bats', 'cats', 'dogs', 'side'] returned |
| 8.2 | # Test exclusion file does not exclude start word  start = 'side'  lines = ['bats', 'cats', 'dogs', 'side']  excluded = ['bats', 'side'] | ['cats', 'dogs', 'side'] returned | ['cats', 'dogs', 'side'] returned |
| 8.3 | # Test exclusion file no matching words  start = 'side'  lines = ['bats', 'cats', 'dogs', 'side']  excluded = ['bots', 'sits'] | ['bats', 'cats', 'dogs', 'side'] returned | ['bats', 'cats', 'dogs', 'side'] returned |

## Unit Test Execution

The following screen shots present script execution of the test\_word\_ladder.py file using a batch script file.







# Requirements Acceptance Tests

The following table lists outcome of Requirements Acceptance Testing undertaken.

| **Software Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
|  | The program shall read in a single “not empty” file input by the user. The program will reprompt until a candidate file is found. This file must have a valid file name and not be empty. The file must be a text file. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall optionally prompt the user to supply a list of excluded words. If the decision is made to provide a list of excluded words then the program will reprompt until a candidate file is found. This file must have a valid file name and not be empty. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall prompt the user to enter a starting word. This word must exist in the input file. The word must be alphabetic and longer than one character. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall prompt the user to enter a target word. This word must exist in the input file and not be an excluded word. The word must be alphabetic and the same size as the start word. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall prompt the user as to whether they want the system to calculate the shortest path. The default value is the longest path. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall display the word path according to the parameters input, together with the size of the path. | Full | Pass | Not Applicable – Test Passed |
|  | The program will read in the list of words and exclude any words that have been provided in the excluded word list. | Full | Pass | Not Applicable – Test Passed |
|  | The program shall find the path from one word to another word by changing a letter at a time. For example, it will find the path of words from hide->side->site->sits->sets->sees->seek. All words used within the word ladder must be contained in the list of words for the game. | Full | Pass | Not Applicable – Test Passed |
|  | If a path from the start word to the target word can not be found then display “No Path Found”. | Full | Pass | Not Applicable – Test Passed |

# User Instructions

Instructions on executing the program are as follows:

1. Run word\_ladder.bat to execute the program
2. User will be prompted to provide the name of a valid dictionary to use for the game. Users will be reprompted until a valid name is provided.
3. User will be prompted to indicate whether to provide a list of words to exclude. If Y, then user must provide a valid file name and will be prompted until they do.
4. User will be prompted to provide a valid start word.
5. User will be prompted to provide a valid target word
6. User will be prompted to indicate whether to provide the shortest path.
7. Path between words will be displayed. If no path found then message indicating this will be displayed.

## Execution example

Following is an execution example using word\_ladder.bat in Windows environment

