# Python Word Ladder Assignment 1

S5002464, s5046838

Blake Godfrey, Kayn Critchley

[blake.godfrey@griffithuni.edu.au](mailto:blake.godfrey@griffithuni.edu.au). [kayn.critchley@griffithuni.edu.au](mailto:kayn.critchley@griffithuni.edu.au).

Github link.

<https://github.com/s5046838/python>

**Contents**

1. Problem Statement 1

2. User Requirements 2

3. Software Requirements 2

4. Software Design 3

5. Requirement Acceptance Tests 5

6. Detailed Software Testing 6

7. User Instructions 7

## Problem Statement

In this assignment we were required to make small groups of two to complete and test a python program code to demonstrate our understanding of the scripting languages taught in the course. We are to upload and edit our python code on the github website to show how has done what and for easier access for the group to edit the code. With the given python code we must complete the program and test existing functions to ensure all code is functioning correctly.

## User Requirements

* The user enter a start word for the word ladder
* The user enter a target word for the word ladder
* The user enters a word to be removed from the word ladder
* The user cannot compare integers, blank words
* The user cannot remove a word that is the start or target word.

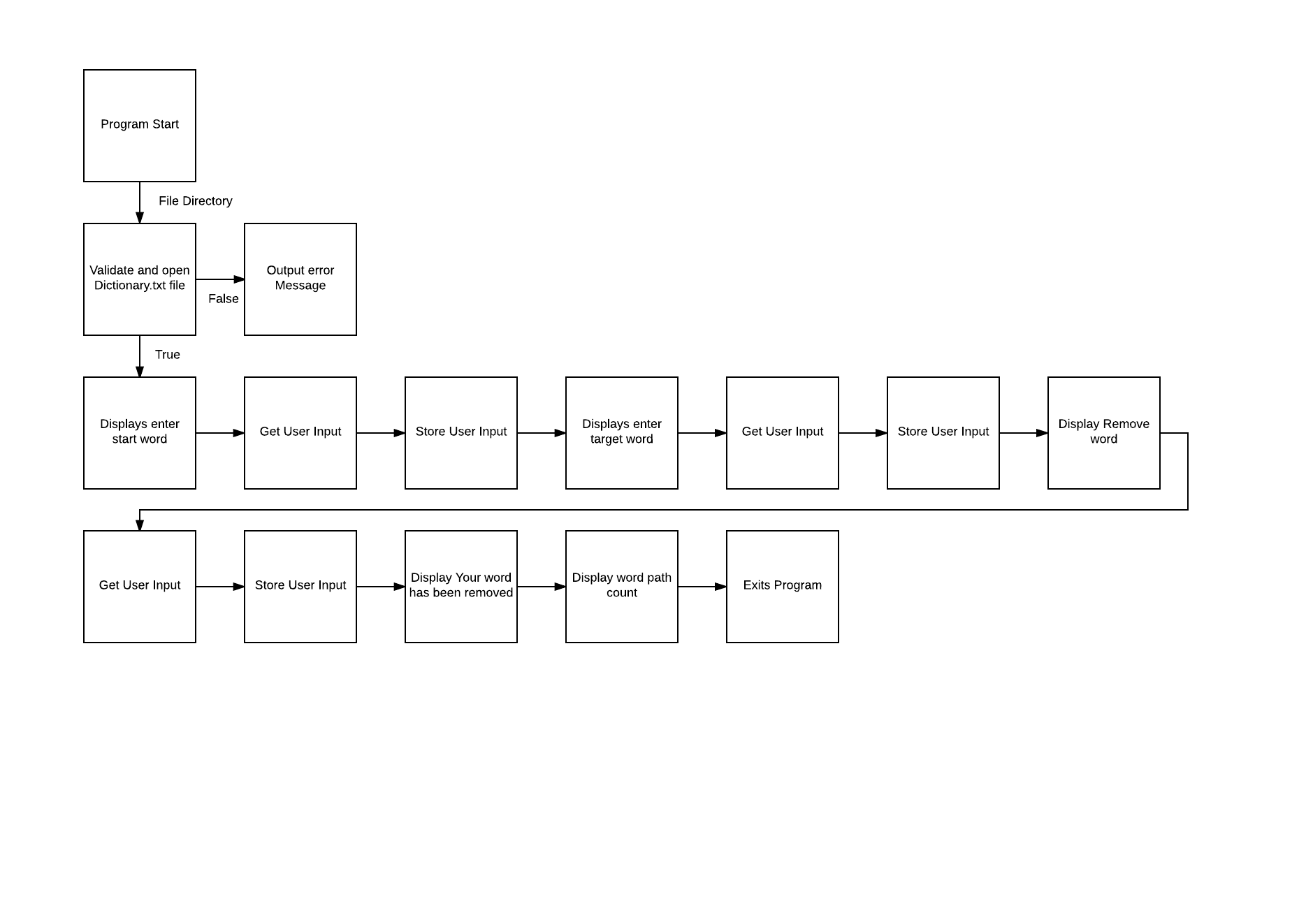
## 

## Software Requirements

* The program imports the RE module for searching
* The program uses a function to read the dictionary.txt file, which contains the entire english dictionary.
* The program reads each word individually in the dictionary.txt file
* The program makes a user’s input for start, target and removed word.
* Prints out if user’s removed word has been removed or not
* Prints out the number of how many words it took to reach the target word
* Prints out the total of how many words it took to reach the target word

## Software Design

### High Level Design – Logical Block Diagram



### List of all functions in the software.######

* File: Opens and reads the dictionary file.
* Same: Checks tuples and uses the zip function to return the list of tuples. Also returns items that are identical in the same position in the list.
* Build: Goes through a search to see if the pattern words match.
* Find:

1. Checks if letters in the appended list match, how many in the word are being checked using the build function and if the length of the list is equal to 2.
2. Goes through the list of tuples in ascending order and then in reverse.
3. Checks if the match is either greater or equal than the length target minus one character.
4. Checks if the match length is the same and appends item into path and returns true.
5. Appends item into path, if any string values are found to match return true, if no string values match remove it from path.
6. Checks if the user’s input is correct and makes sure the chosen start or target word aren’t removed.
7. Checks if user’s input is numerical.
8. Checks if user’s input is empty.
9. Checks if user’s input is the same length.

* WordRemove: Removes words from the list of words.

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

For each data structure in the list the following information is provided:

* List = [] : Stores the letters of the word to be checked and replaced
* Word = [] : Stores the start, target word from the dictionary that will be used
* Path = [start] : Stores the start word as an index
* Path= {start: True} : Stores the start word as a key and gives it the value of True

### Detailed Design – Pseudocode for all non-standard and non-trivial algorithms that operate on datastructures

import re # Re is a module for string searching

* **Function for opening the file**

Define function file

Open the file and read

Read each line separately

Then return each word separately

The dictionary text file will be read

* **Function for zipping tuples and returning their values**

Define function same

def same(item, target):# checks tuples and uses the zip function to return the list of tuples

return len([x for (x, y) in zip(item, target) if x == y]) # Returns items that are identical in the same position in the list

* **Function for searching if string values are in the given parameters**

def build(pattern, words, seen, list):

return [word for word in words

if re.search(pattern, word) and word not in seen.keys() and #Goes through and searches if pattern matches word

word not in list]

* **Finds words, letters in the given path**

def find(word, words, seen, target, path): # Function for checking if letters in the appended list match

list = [] is the empty array

for each value in the range of the length of word

list equals(list plus word index position) plus (word index position plus 1) and words, seen and list.

If length of list returns true on a length of two

Return false to the function

List is equal to a sorted array of the same items and target values for item in the list

List is then reverse

For each match and item in the list

If match is greater or equal to length of target's character minus 1 or match returns the same length as target minus 1 as True.

Function returns true and appends item into path

Seen list with the index of item equals True

For each match and item in the list:

Append item to the path list

If item, words, seen, target or path are found

Return True and remove the value from the path

* **A while True statement for user error checking**

while True: # This is used or checking if the users input is correct

start = input("Enter start word:") # Initilizing word

target = input("Enter target word:") # finalized word

remove = input("Removed a word:") # word you want removed

words = [] is the empty array

If the word to be removed is equal to the user’s start word or target word

Comes up with an error saying that those words cannot be removed and restarts from the beginning

If the user’s start, target or removed input word is a number:

Comes up with an error saying that those numbers cannot be compared and restarts from the beginning

If the user’s start, target or removed input word is empty:

Comes up with an error saying that there’s an empty value and restarts from the beginning

If length of start word and length of target word are equal:

For each line in lines:

Word is equal to the line being stripped

Word is then appended

It then breaks

Else print an error

* **A few arrays and if statement for displaying the end results**

path = [start] # The starting word used to create the list of words

seen = {start: True} # The dictionary key, value stored if it is True

if find(start, words, seen, target, path): # Shows the total number of words used and the words used to change your start word to the target word.

path.append(target)

else:

print("No path found")

* **Function for removing a word from the path**

Def function wordremove.

If the words in the path are not equal to first index in the path array.

Remove the word from the path

Return text saying “Your word has been removed”

Or

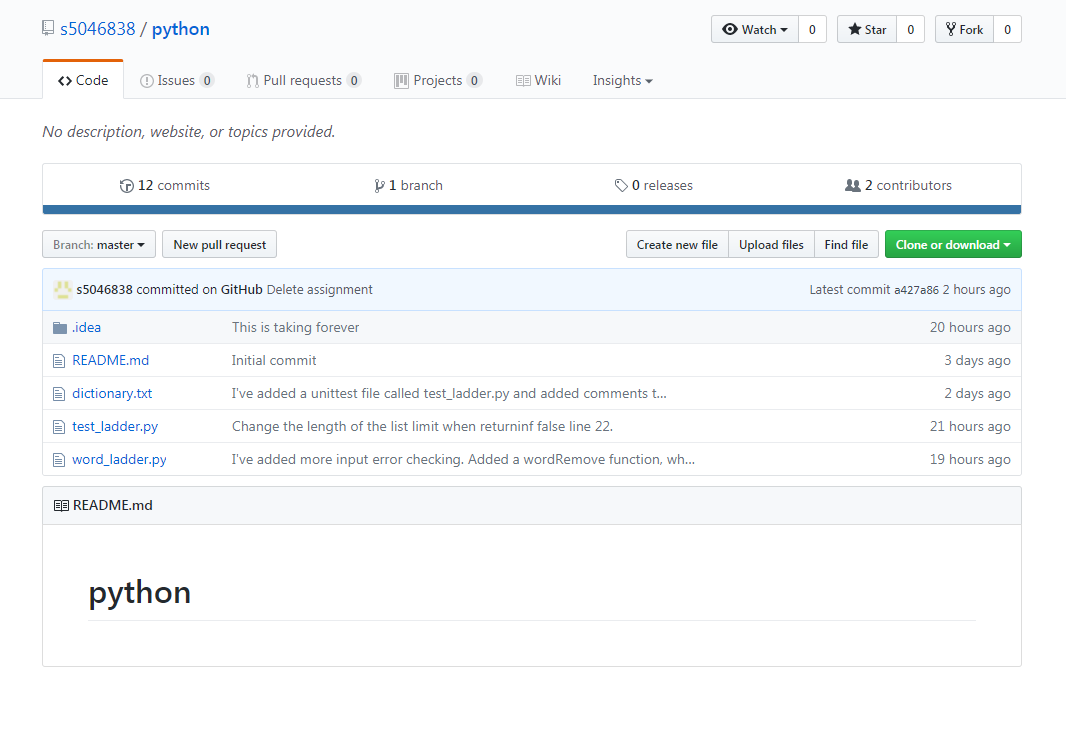
Return text saying “Your word was not in the path”

print(wordRemove(path, remove) + "\n","Path count :", (len(path) - 1), "\n", "Words in

### Configuration management and version control

## We created a repository on Kayn’s github account.

1. Using git CMD, we both cloned the repository to our computers.
2. Using add, we added the Python word\_ladder.py, test\_ladder.py and dictionary.txt file
3. When work had been done on our computer, which was ready for upload. We would commit the changes with a comment explaining them.
4. After committing, we would push the commit onto github.
5. Then if need be, we would pull any changes from github onto our computer for us to look at.



## Unit Tests

|  |  |  |
| --- | --- | --- |
| **No** | **Test Case** | **Expected Results** |
| 1 | Tests if The File Works | Returns true if the file works fine. |
| 2 | Tests if File Doesn’t Work | Returns true if the file doesn't work. |
| 3 | Test if Wrong file is being read | Returns true if there is no file. |
| 4 | Tests if Word Inputs Work | Returns true if the word inputs are working. |
| 5 | Test if Word Inputs don’t Work | Returns true if the word inputs don’t work. |
| 6 | Tests if Removed Word is in path | Returns true if the removed word is in the path. |
| 7 | Tests if Removed Word isn't in Path | Returns true if the word is in the path. |
| 8 | Tests if Starting Word is equal to the word user wants to be removed | Returns true if the starting word is equal to the wanted removed word. |
| 9 | Tests if the User input is null | Returns true if the user doesn’t input a word. |

## Requirement Acceptance Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SoftwareRequirement No | Test | Implemented (Full /Partial/ None) | Test Results (Pass/ Fail) | Comments (for partial implementation or failed test results) |
| 1 | Program accepts the dictionary text file. | Full | Pass |  |
| 2 | Program does not allow you to compare integers | Full | Pass |  |
| 3 | Program does not allow you to compare empty spaces | Full | Pass |  |
| 4 | Program does not allow you to compare words of different length | Full | Pass |  |
| 5 | Program allows you to remove a word from the list | Partial | Pass | It removes a word without problem, but cannot remove multiple words |
| 6 | Program goes from hide to seek in 6 steps | Full | Pass |  |
| 7 | Program goes from seek to hide in 6 steps | partial | Pass | It does the opposite in 10-12 steps. |
| 8 | User has the choice of asking for the shortest route | Full | Pass |  |
| 9 | User cannot remove a start or target word | Full | Pass |  |

## 

## User Instructions

* Open and extract the zip file.
* Open the word doc in the zip file.
* Click on the Github link to our assignment.
* Open the Word\_ladder.py link.
* Run Word\_laddy.py in pycharm.
* Enter in a start word and press enter.
* Enter in a target word. (Must have the same number of words as the start word.)
* Enter a word to remove.
* Run it
* Get results