**WORDZONE**

*Submitted in partial fulfilment for the award of the degree of*

**Bachelor of Technology**

in  
**Information Technology**

*by*

**RITVEAK DUGAR**

**16BIT0407**

**Under the guidance of**

**Prof. E Sathiyamoorthy**

**School of Information Technology & Engineering**

**VIT, Vellore.**



March,2020

**DECLARATION**

I hereby declare that the thesis entitled “WORDZONE ” submitted by me, for the award of the degree of Bachelor of Technology in Information Technology to VIT is a record of bonafide work carried out by me under the supervision of Prof E Sathiyamoorthy.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: Signature of the Candidate

**CERTIFICATE**

This is to certify that the thesis entitled “WORDZONE**”** submitted by RITVEAK DUGAR (16BIT0407) SCHOOL OF INFORMATION TECHNOLOGY & ENGINEERING VIT, for the award of the degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by him under my supervision during the period, 01. 12. 2019 to 30.04.2020, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted  
either in part or in full, for the award of any other degree or diploma in this institute or  
any other institute or university. The Project report fulfils the requirements and regulations of  
University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : **Signature of the Guide**

**Internal Examiner**  **External Examiner**

Head of the Department

SITE

Date: …………….…..

**CERTIFICATE BY THE EXTERNAL GUIDE**

This is to certify that the project report entitled “**WORDZONE”** submitted by R**ITVEAK DUGAR (16BIT0407)** to Vellore Institute of Technology in partial fulfilment of the requirement for the award of the degree of BTECH in Information Technology is a record of bonafide work carried out by him under my guidance. The project fulfils the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Naveen Chandra**

**EXTERNAL SUPERVISOR**

**Product Engineer**

IDOS India Pvt Ltd

Bhive, Mahalakshmi Chambers, #29, MG Road, Bengaluru, 560001

+91 8496816918

[naveen@myidos.com](mailto:naveen@myidos.com)

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude to Prof E Sathiyamoorthy, Professor Grade 1, School of Information Technology and Engineering, Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavour.

I would like to express my gratitude to Dr.G.Vishwanathan, Mr. Sankar Viswanathan, Dr. Anand A. Samuel, Dr. S. Narayanan, and Dr Balakrushna Tripathy, School of Information Technology and Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to <Program char-name>. <Program Chair and designation>, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: **Ritveak Dugar**

**ABSTRACT**

This project is an attempt to bring in various word related paradigms in a single place, where users can learn, understand and find their words. The project has four main sections namely Find Zone, Understand Zone, Knowledge Zone and Learning Zone.

Find Zone is the section where user can find words they want, for example, rhyming words with particular meaning or a word for their crossword with particular hints or words for their scrabble game.

Understand Zone is the section where users can understand a word by looking up multiple meanings and examples of it.

Knowledge Zone is the section where users can know more related words like synonyms, antonymns, hypernymn, holonymn and hyponymn.

Learning Zone is the section which utilises the searches done in the whole application to train a machine learning model. This model smartly suggests words for the user to learn.

The whole project is aimed to improve vocabulary of users by giving them a platform to find, understand, know and learn words based on their personalised usage.

**CONTENTS**

1. **– INTRODUCTION**
   1. OBJECTIVE 9

1.2 MOTIVATION 9

1.3 COMPONENTS 9

**2 – LITERATURE SURVEY 11**

**3 – DESIGN**

3.1 UML DESIGN 13

3.1.1 FIND ZONE 13

3.1.2 KNOWLEDGE ZONE 14

3.1.3 UNDERSTAND ZONE 15

3.1.3 LEARNING ZONE 16

3.2 SYSTEM ARCHITECTURE 17

**4 – SOFTWARE REQUIREMENTS**

4.1 PROGRAMMING LANGUAGE 18

4.1.1 PYTHON 18

4.1.2 KIVY 18

4.2 LIBRARIES 19

4.2.1 NLTK 20

4.2.2 SPELL CHECKER 21

4.2.3 PRONOUNCING 21

4.2.4 DATAMUSE AND OXFORD API 24

4.2.5 WORDFREQ 24

4.2.6 PICKLE 24

4.3 TRAINING MODEL 19

4.3.1 LSTM 20

**5 – MODULE DESCRIPTION**

5.1 FIND ZONE 25

5.1.1 RHYMING ZONE 25

5.1.2 CROSSWORD HELPER 30

5.1.3 SCRABBLE HELPER 35

5.2 UNDERSTAND ZONE 40

5.3 KNOWLEDGE ZONE 41

5.3.1 SIMILAR MEANING WORDS 42

5.3.2 OPPOSITE MEANING WORDS 44

5.3.3 HYPERNYM / HYPONYMN / HOLONYM 46

5.4 LEARNING ZONE 48

5.4.1 LOW LEVEL WORDS 49

5.4.2 MEDIUM LEVEL WORDS 49

5.4.3 HIGH LEVEL WORDS 49

5.4.4 TRAINING AND PREDICTION 50

**Chapter 1**

**Introduction**

* 1. OBJECTIVE

The objective of WORDZONE is to provide a single platform for users to do multiple word related searches and improve their vocabulary by learning words that are suggested to them based on their personalised usage.

Therefore giving a wide range of functionalities to the user to play around words and increase their knowledge as well.

1.2 MOTIVATION

Being a writer I often feel the need to improve my vocabulary and at times look for specific words as well. However increasing one’s vocabulary is a tough task to do, because there is no magical/efficient place where you can get words on the basis of your own vocabulary.  
While studying Machine Learning, an idea hit my mind: What if a model is trained on basis of my word searches and suggests me new words to learn on the basis of that!

Hence to accomplish it, WORDZONE has a lot of word search related sections, which enables user to not only find the words they are looking for but passively generate personalised set of word data as well.

1.3 COMPONENTS

Wordzone has four main sections, which combined makes wordzone a one of its own platform:

1. Find Zone
2. Understand Zone
3. Knowledge Zone
4. Learning Zone

1.3.1 FIND ZONE

FIND ZONE allows the user to find words that they are looking for.

A user can look up for words which rhyme with a given word, further the result can be narrowed down to be of some specific meaning in the “Rhyming Words” section.

User can also look up for words with some meaning, which can be narrowed down to a list of words with fixed character at a position in the “Crossword Helper” section.

Words for Scrabble can be searched using “Scrabble Helper” where the word fragment, its position and length of the word can be found.

1.3.2 UNDERSTAND ZONE

UNDERSTAND ZONE allows user to find the meaning and examples of the word they enter. It also shows multiple meanings that a word might have and returns example for each case as well.

1.3.3 KNOWLEDGE ZONE

KNOWLEDGE ZONE allows user to know more words around the word they enter. A user can find Synonyms in the “Similar meaning” section, antonyms in “Opposite meaning” section and other related words in “Hypernymn/Hyponymn/Holonymn” section

1.3.4 LEARNING ZONE

LEARNING ZONE is the section that makes use of all the searches made by the user in the application to smartly suggest new words to learn. The searches and results help to train three different models, giving user option to learn a low-level, mid-level and high-lever word. The model can be retrained manually, whenever substantially new data is being recorded. Hence making WORDZONE an evolving model which not only gives user a platform to find, understand and know words, but makes their learning personalised as well.

**Chapter 2**

**Literature Survey**

A lot of projects have been made previously on NLP using the features, tools and techniques that are used in wordzone.  
Below is a literature survey on few of those tools, techniques and properties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ref no.** | **Topic** | **Summary** | **Use/Meaning** |
| 1. | Web Corpus[1] | Traditionally written corpora are primarily recorded from print media, with advent of internet, web data can be used to train corpora as well. | Corpora are basically huge collections of words and their associated features like meanings, pronunciations. They are used for NLP based usages |
| 2. | NLTK[2] | The need of an umbrella covering a lot of linguistics related paradigm gave birth to NLTK toolkit. | NLTK being a collection of modules which helps in computation of linguistics is extensively used in projects related to NLP. |
| 3. | WordNet[3] | WordNet is a lexical database for the English language. It is basically a combination of dictionary and thesaurus which contains words, their definitions and many other related words. | Wordnet acts as database which is used for fetching properties of words and their related words. |
| 4. | Semantic Analysis[4] | Latent Semantic Analysis is a good approach for finding accurateresults. Another important use of LSA is to find out semantic similarities between different set of textual data. | Semantic Analysis helps in determining the meaning of sentences. |
| 5. | Phonetics[5] | Phonology is used to recognize sound so as to find out the language, semantic and syntactic meaning. It also helps in pronouncing words and finding phonetically similar words. | Stores data related to pronunciation of words, hence used in projects related to rhyming words. |
| 6. | Python GUI [6] | There are many GUI libraries forPython programs. The most famous ones are: Kivy,Tkinter, PyQt, PyGUI,Pyforms, PySide, Flexx, PySimpleGUI, IPyWidgets, Wax Python GUI, etc. | To give a Graphical User Interface to the python programs. |
| 7. | Dictionary[7][8] | Dictionary requires corpus which contains words and its attributes like meaning, pronunciation, example etc. so that it can also be used for machine translation and language processing. | In addition to using dictionary for finding meanings, one can also use it to find rhyming words and pronunciations. |
| 8. | LSTM[9] | LSTM is a Machine Learning algorithm used for efficiently predicting and training on time series data. | LSTM can be used for predicting the word based on the data generated by the user’s usage. |

**Chapter 3**

**Design**

Design is an integral part of development, it paves the pathway for the development.

Below are the designs made for wordzone.

3.1 UML Diagrams

3.1.1 Find Zone:



3.1.2 Understand Zone:

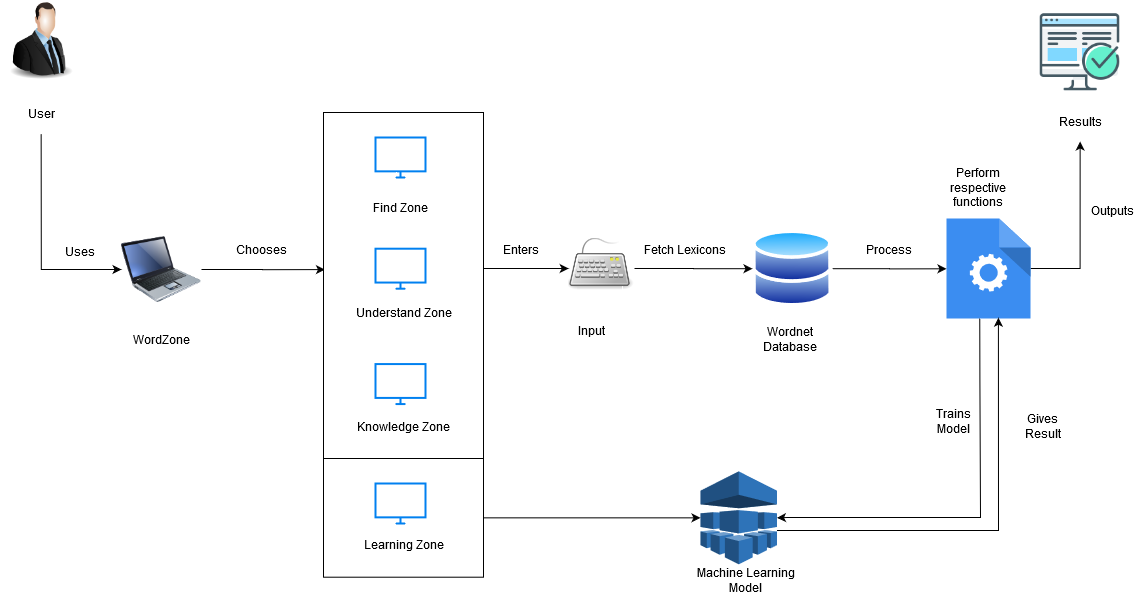
3.1.3 Knowledge Zone:

****

3.1.4 Learning Zone:

****

3.2 System Architecture

****

**Chapter 4**

**Software Requirements**

4.1 PROGRAMMING LANGUAGES

Programming language is the base for any software development; they act as the field where the crop grows. Hence there is always a particular kind of field which is best for a given crop.

The following are the programming languages that are used in WORDZONE.

4.1.1 PYTHON

Python is a high level programming language which is always preferred over others when it comes to Machine Learning and Natural Language Processing, because of the wide variety of libraries that it supports.

Since WORDZONE uses NLP and ML, python was the go to language.

Python being easy and widely used has a lot of support online, hence development and troubleshooting is eased because of the rich documentation and forums online.

Many libraries like NLTK, pronouncing, spellchecker allowed implementation and raw data for processing in the software.

The function written in python then processes the data and returns result.

Python in WORDZONE is primarily used for the all round processing, and hence it can truly be called as the soul of wordzone.

This soul however needed a body as well, so to provide a Graphical User Interface, Kivy was selected.

4.1.2 KIVY

Kivy is an open source python framework for developing user interfaces. There were other options for GUI like Tkinter, Pyqt and many others. Kivy stood out in the line because the modern look and the fact that it is more customisable.

Kivy has its own syntax for defining screens, buttons and all other graphical components. Thereby aiding in development of rich but simplistic User Interface.

4.2 TOOLS AND LIBRARIES

There are few tools and libraries which helped in building WORDZONE apart from the basic programming languages. All these tools and libraries are freely available and have good documentation to understand and work with them. There are the common ones like Pandas, Numpy, Keras, Tensorflow, Pickle and other specific ones like NLTK, Pronouncing, Spell Checker,wordfreq; all these along with logic facilitate the working of wordzone.

4.2.1 NLTK

NLTK – Natural Language Toolkit is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language. This is one of the most usable and mother of all NLP libraries.

It consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, pre-process, and understand the written text.

Since wordzone only plays around words and not the syntactic meanings, wordnet is used at most. Wordnet is a lexical database which records semantic relation of words including synonyms, hyponyms, and meronyms. The synonyms are grouped also known as synsets with short definitions and usage examples. This comes in handy for making a rich group of related words.

Wordnet acts as a corpus, which is nothing but a collection of words. In the view of wordzone, wordnet acts as our database which has a lot of words along with some associated meaning and properties. The words are in turn linked to each other forming a network of words which helps to find related words to a given word.

The logic whatsoever required is applied on data collected from wordnet which can be made rich by including more and more related words.

For example in Rhyming words section, where the rhyming word is required to be related to something, a rich collection of words are made by using wordnet:

syns=wordnet.synsets(mword)

        for syn in syns:

            rr+=syn.lemma\_names()

        #Since simple synonym set is not enough, lets add more related words

        #finding all related words among which rhyming words is to be found

        hr=[]

        syns=wordnet.synsets(mword)

        for syn in syns:

            sn=syn.hypernyms()#broader category:colour is a hypernym of red.

            an=syn.hyponyms() #narrower category - red : color

            dn=syn.member\_holonyms()#Body is a holonym of arm, leg and heart

            for s in sn:

                hr+=s.lemma\_names()

            for a in an:

                hr+=a.lemma\_names()

            for d in dn:

                hr+=d.lemma\_names()

        #now even "loaf" gets included when "food" is given as input

        #making the list richer by adding synonyms of the words which are in hr.

        fn=[]

        for h in hr:

            ss=wordnet.synsets(h)

            for s in ss:

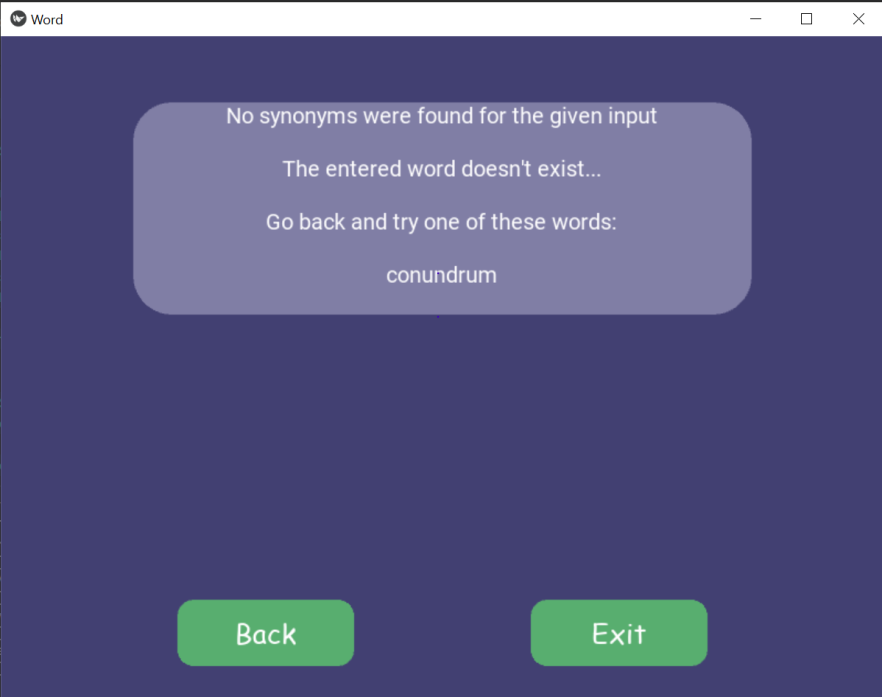
                fn+=s.lemma\_names()

Thereby making the collection rich and then applying logic.

4.2.2 SPELL CHECKER

Spell checker is a library in Python which suggest words near the wrongly spelled words. This helps in the result pages when there is no result due to wrong spelling.

Below is an example where the entered word was “Conundrom” instead of “Conundrum”:



Code snippet:

        if(len(synonyms)>0):

            ss=set(synonyms)

            new = ""

            for x in ss:

                new += x+"\n"

            return new

        else:

            str="No synonyms were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

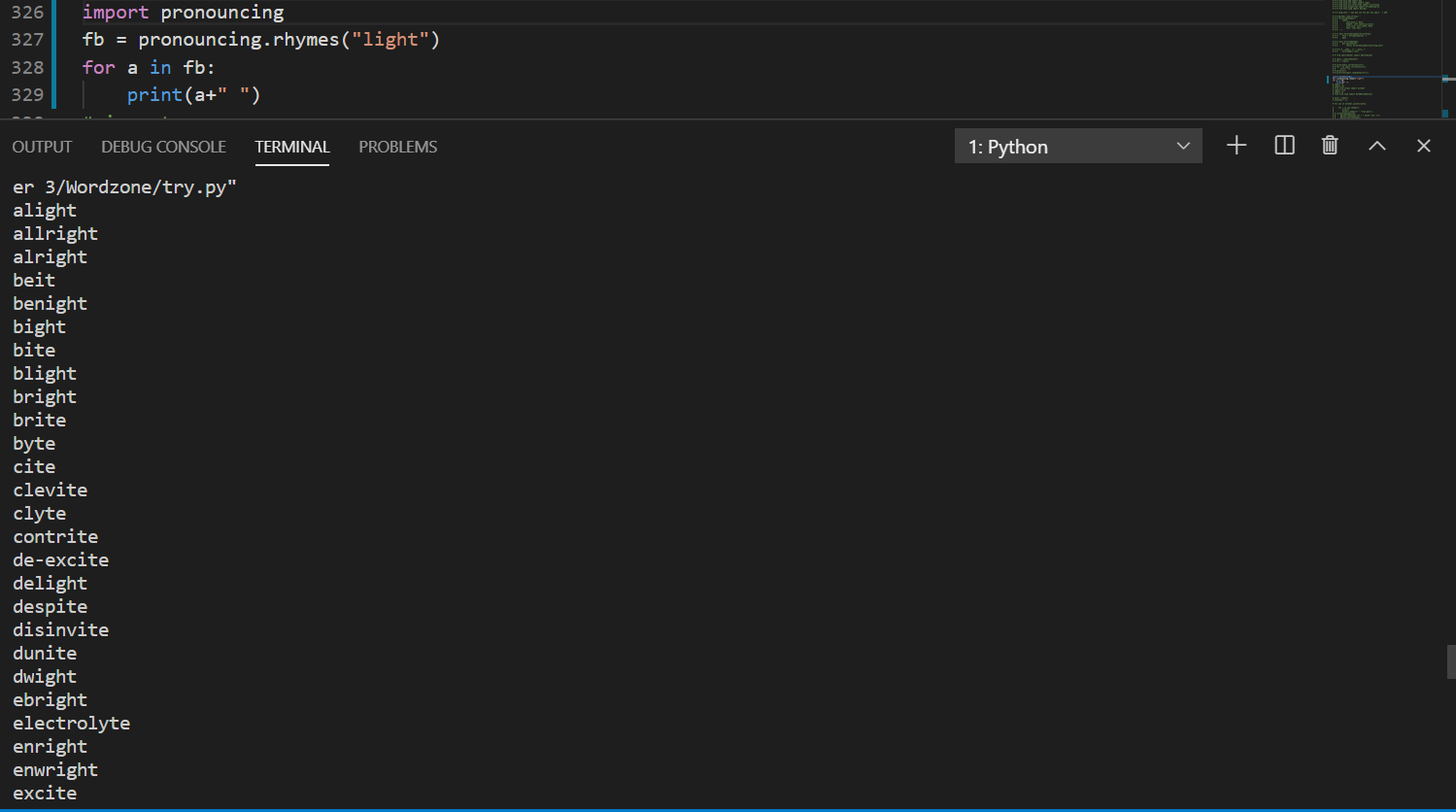
                    str+=s+"\n"

            return str

4.2.2 PRONOUNCING

Pronouncing is a library in python which helps in finding similar sounding words, it makes use of CMU Pronouncing Dictionary which holds the phonetics of words.

For example, searching rhyming words to “Light” would return:



In WordZone, it is used as a part of rhymer dictionary:

def rhy(self,word):

        if(word==""):

            return "No word entered"

        fb = pronouncing.rhymes(word)

        if(len(fb)>0):

            ss=set(fb)

            new = ""

            for x in ss:

                new += x+"\n"

            #print(new)

            return new

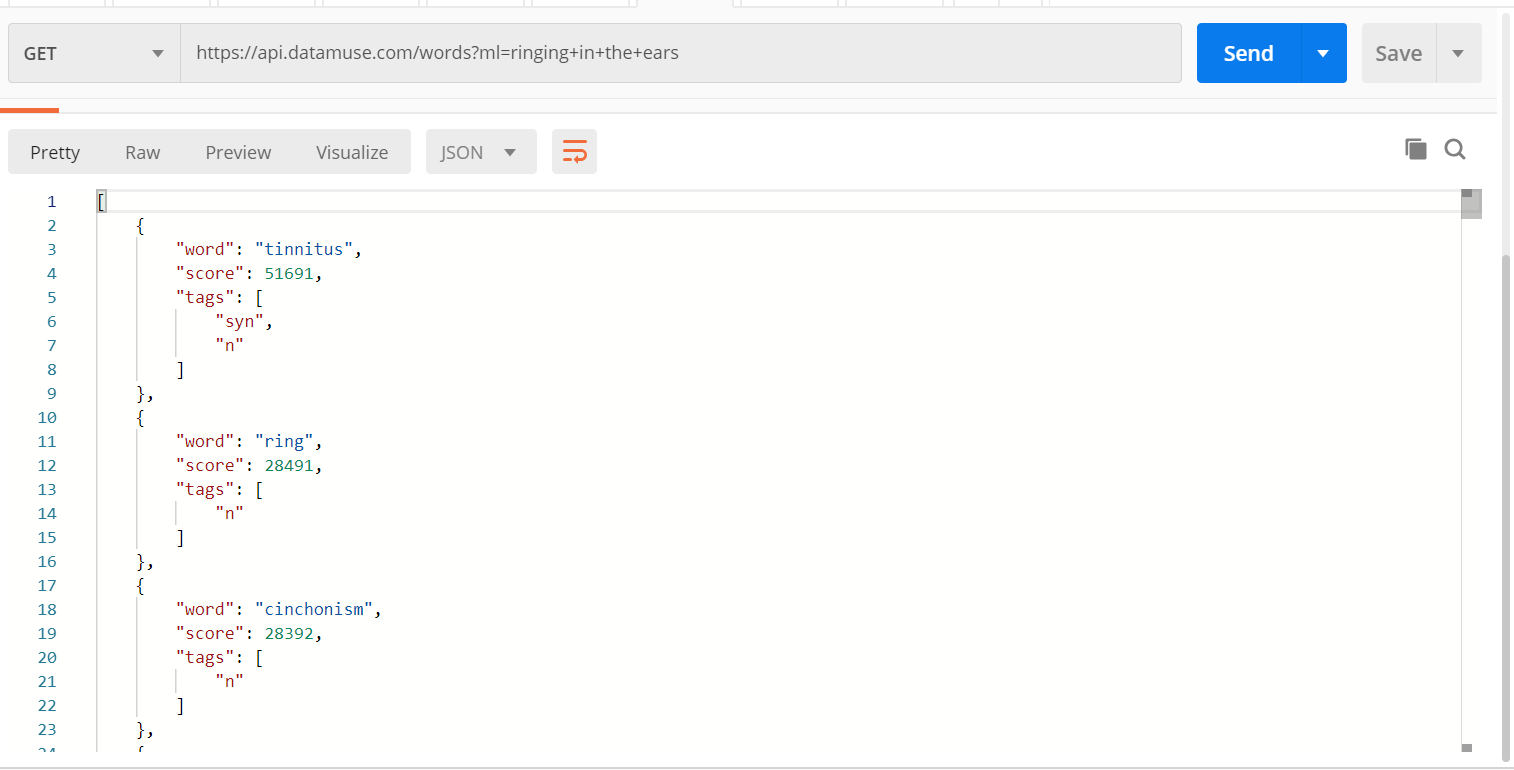
4.2.4 DATAMUSE AND OXFORD API

The Datamuse API is a word-finding query engine which has a lot of methods.

NLTK is a more raw form of receiving data, but Datamuse is a smarter way.

One can use it to find words that match a given set of constraints.

For example searching for words related to a particular description let’s say“Ringing in ears” would give:



This is used in Crossword Helper, where we need list of words when a user enters a hint.

Further operations are done on the list obtained by using this API.

Code Snippet:

fn=[]

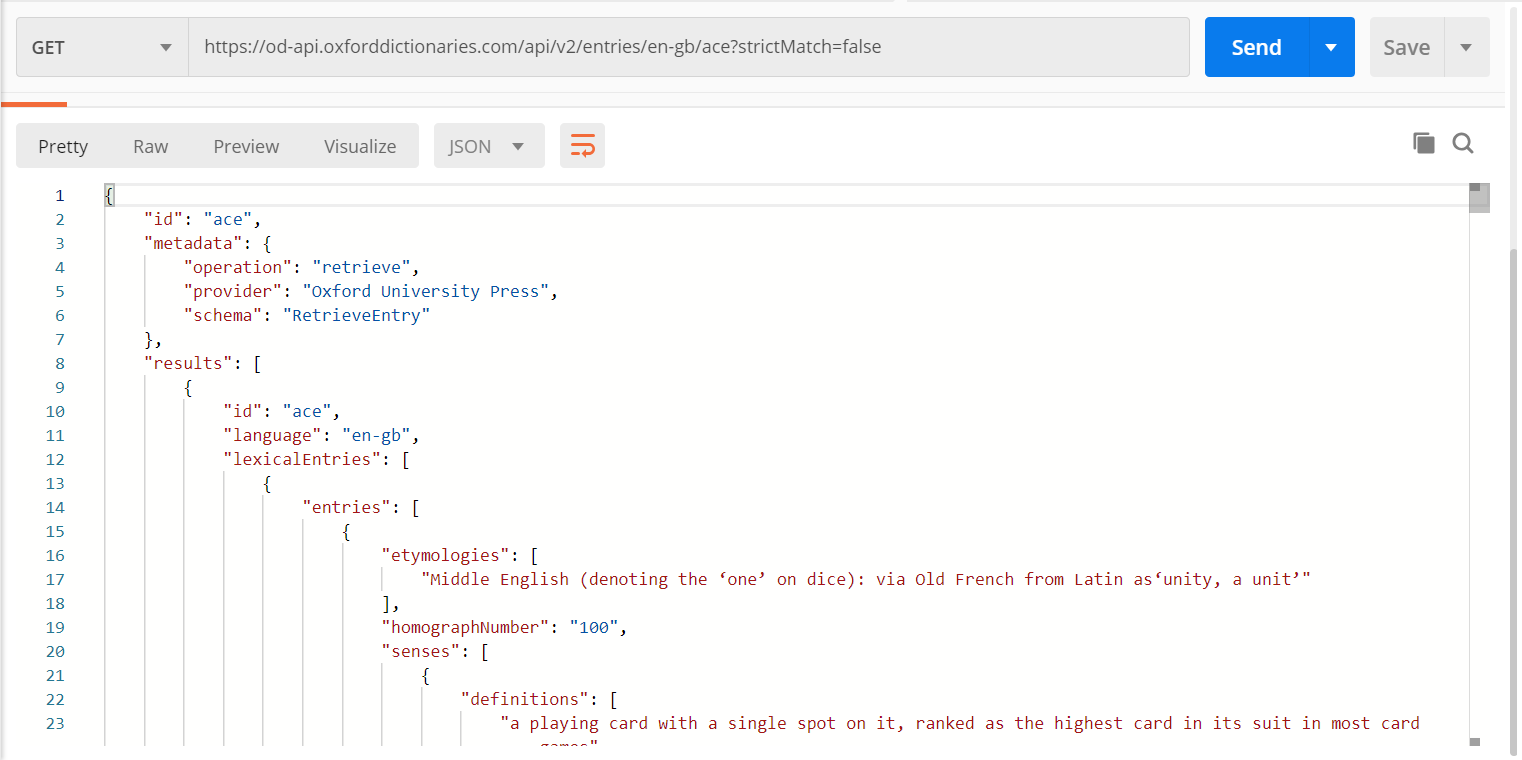
        res= requests.get("https://api.datamuse.com/words?ml="+w)

        for a in res.json():

            fn.append(a["word"])

        #The words related to description are stored in fn now.

The Oxford Dictionary API uses official oxford dictionary for returning word meanings.

For example, it returns all the possible meanings with examples:

This is used in Understanding Zone where user enters a word and gets meanings of it with examples.

Code Snippet:

 str=""

        response= requests.get("https://od-api.oxforddictionaries.com/api/v2/entries/en-gb/"+word+"?strictMatch=false",headers={"Accept": "application/json","app\_id": "c1498ba3","app\_key": "ec959282e97d787344cbe7cfeb13c965"})

        try:

            for a in response.json()["results"]:

                for b in a["lexicalEntries"][0]["entries"][0]["senses"]:

                    str+="Definition:\n "+b["definitions"][0]+"\n"

                    str+="Example: \n"+b["examples"][0]["text"]+"\n\n"

        except KeyError:

            str="No meanings were found for the given inputs\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

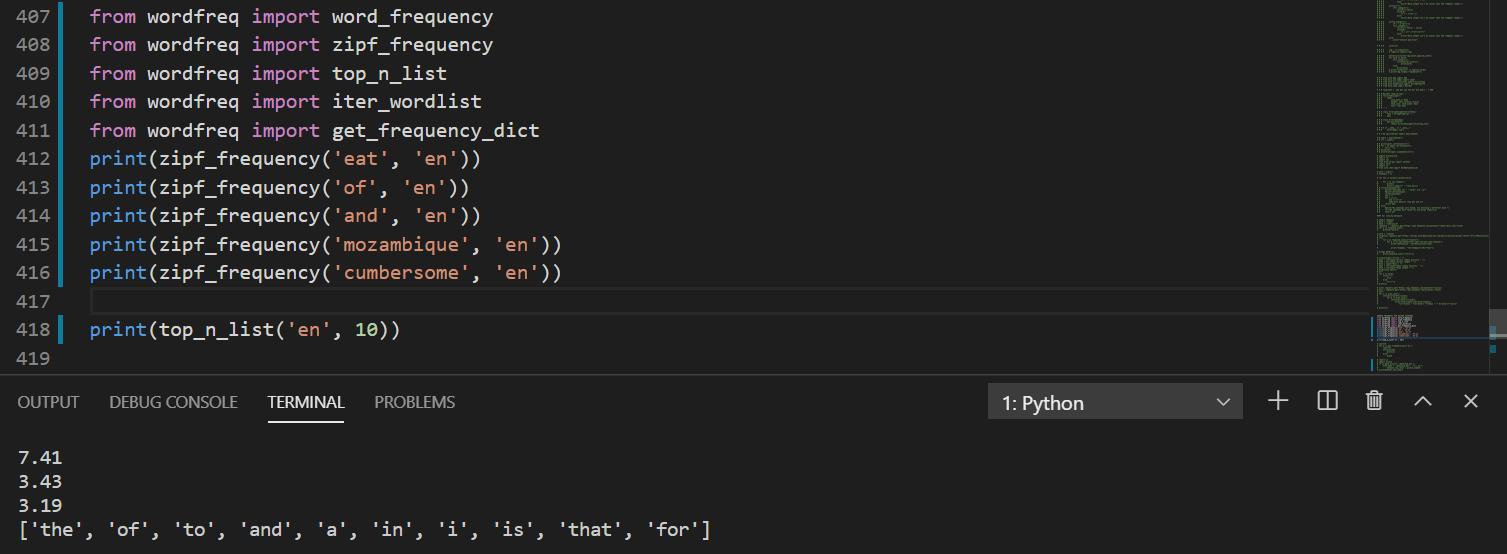
                    str+=s+"\n"

        return str

4.2.5 WORDFREQ

Wordfreq[10] is a python library which gives out frequency of a word in English language. This frequency can be taken as a measure of how common or less common is a word in English language, thereby can be used to show easy (ie, very common) and medium(ie. Common) and hard (ie. Less common) words.

This feature is used in training and predicting words for Leaning Zone which uses the word entered by user in whole wordzone to train a model and then suggesting new words to learn based on the user’s usage.

A demo of what this library returns is below:

4.2.6 PICKLE

Pickle[11] is a library in python which converts data into byte stream(called serializing) and converts byte stream into data (called de-serializing). Pickling is required to store python objects, list in our case.

The data entered by the user in wordzone is of great importance and can be used in suggesting new words to learn, hence pickling comes in handy for storing the data.

In Wordzone, there are three different .pkl files which contains low, medium and high level words based ontheir frequency in English language.

Code snippet:

        if os.path.exists('./data/mid.pkl'):

            with open ('./data/mid.pkl', 'rb') as f:

                dataset\_mid\_level = pickle.load(f)

        else:

            dataset\_mid\_level = []

        if(zipf\_frequency(word, 'en')>2.5 and zipf\_frequency(word , 'en')<4.5):

                dataset\_mid\_level.append(zipf\_frequency(word, 'en'))

        with open ('./data/mid.pkl', 'wb') as f:

            pickle.dump(dataset\_mid\_level, f)

4.3 TRAINING MODEL

Where there is data, there must be a model that trains on it and gives features for predicting upcoming data.

Since WordZone works with the words and inputs given by the user, it receives a lot of data. This data can be used to give a personalised feel to the user in another zone called Learning Zone.

The Leaning Zone basically shows new words on the basis of the user’s past searches.

There are a lot of models that one cana used, but the reason LSTM was chosen is that it is designed to store time series data.

Since user’s vocabulary would be increasing with time, it would be reflected in their usage, which can be recorded and used to predict future values and hence help the user to increase their vocabulary to higher levels.

This requires Time Series Forecasting.

4.3.1 LSTM

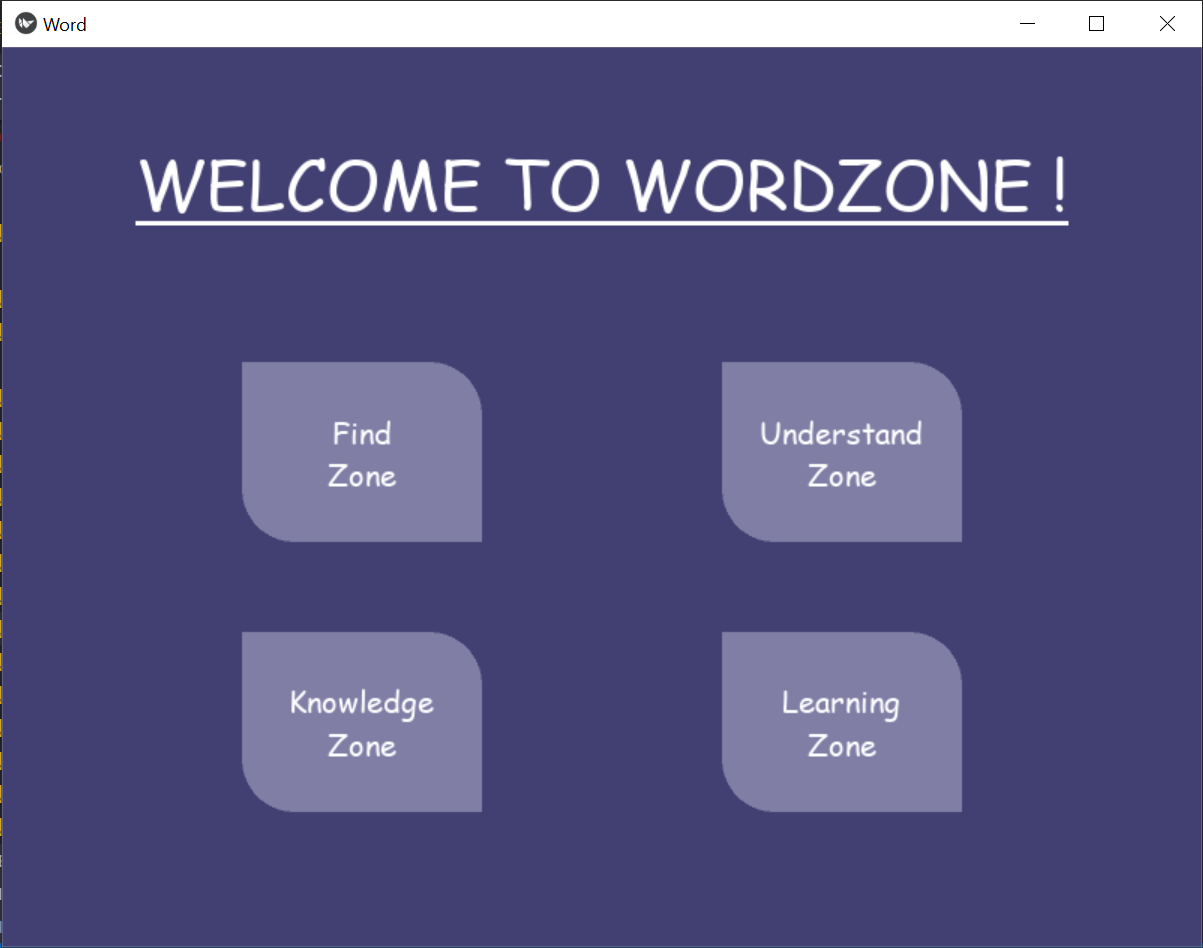
The data generated in WordZone is univariate and over a period of time the user’s vocabulary increases, hence it is a time series data as well.

LSTM is a Machine Learning algorithm which trains on time series data and it works on univariate data as well. Hence LSTM stood as a viable choice for predicting the word based on the univariate time series data generated by the user’s usage.

**Chapter 5**

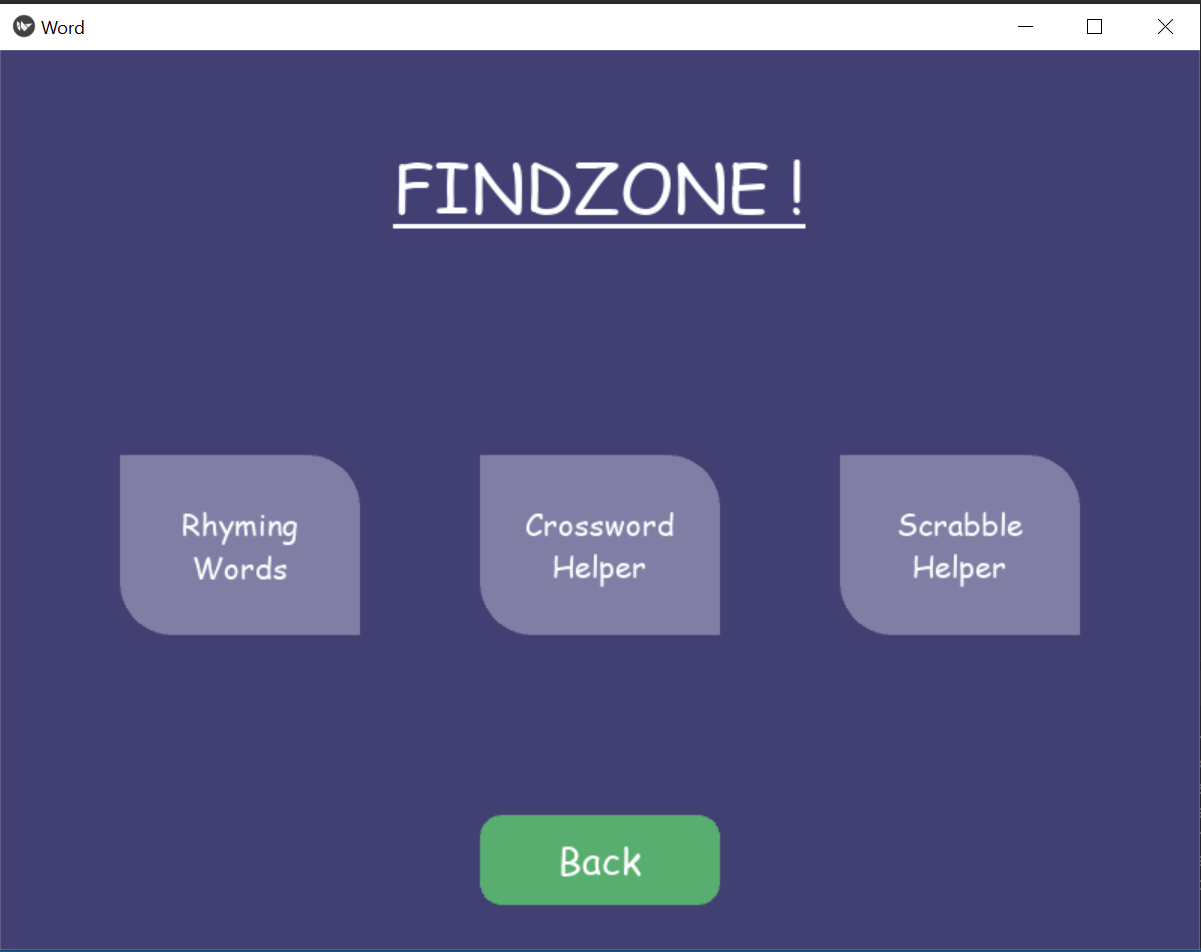
**Module Description**

WORDZONE has four main modules namely - FIND ZONE, UNDERSTAND ZONE, KNOWLEDGE ZONE and LEARNING ZONE. All these modules further have different sections. These sections together contribute to train a model which helps in personalised word suggesting model which helps the user to learn new words.



5.1 FIND ZONE

Find Zone comprises of sections which helps the user in finding words with constraints which belong to either of the three sections:



5.1.1 RHYMING ZONE

Who doesn’t like to put a bit of rhyme into their writings once in a while! This is the section where user can find words rhyming to a particular word, additionally; the user can input a word with which the rhyming word should be related to, thereby easing the way for poets who are looking for a rhyming word with a particular word but with a particular meaning.

Logic Code with explanation in comments:

def rhymerdic(self,iword,mword) :

        if(iword==""):

            return "No word entered"

        # a proper module separately made for finding rhyming words,

based on cmudict.

        fb = pronouncing.rhymes(iword)

        #Simple synonym set

        rr=[]

        syns=wordnet.synsets(mword)

        for syn in syns:

            rr+=syn.lemma\_names()

        #Since simple synonym set is not enough, lets add more related words

        #finding all related words among which rhyming words is to be found

        hr=[]

        syns=wordnet.synsets(mword)

        for syn in syns:

            sn=syn.hypernyms()#broader category:colour is a hypernym of red.

            an=syn.hyponyms() #narrower category - red : color

            dn=syn.member\_holonyms()#Body is a holonym of arm, leg and heart

            for s in sn:

                hr+=s.lemma\_names()

            for a in an:

                hr+=a.lemma\_names()

            for d in dn:

                hr+=d.lemma\_names()

        #now even "loaf" gets included when "food" is given as input

        #making the list richer by adding synonyms of the words which are in hr.

        fn=[]

        for h in hr:

            ss=wordnet.synsets(h)

            for s in ss:

                fn+=s.lemma\_names()

        fn = list(dict.fromkeys(fn)) # removing duplicates

        #now selecting only the words that are common in both

        fo = list(set(fb)&set(fn))

        for chk in fb:

            for chk1 in fn:

                my\_regex = r".\*" + re.escape(chk) + r"$"

                found =(re.search(my\_regex ,chk1, re.M|re.I))

                if found:

                    fo.append(found.group())

        fo=list(set(fo))

        new = ""

        for x in fo:

            new += x+"\n"

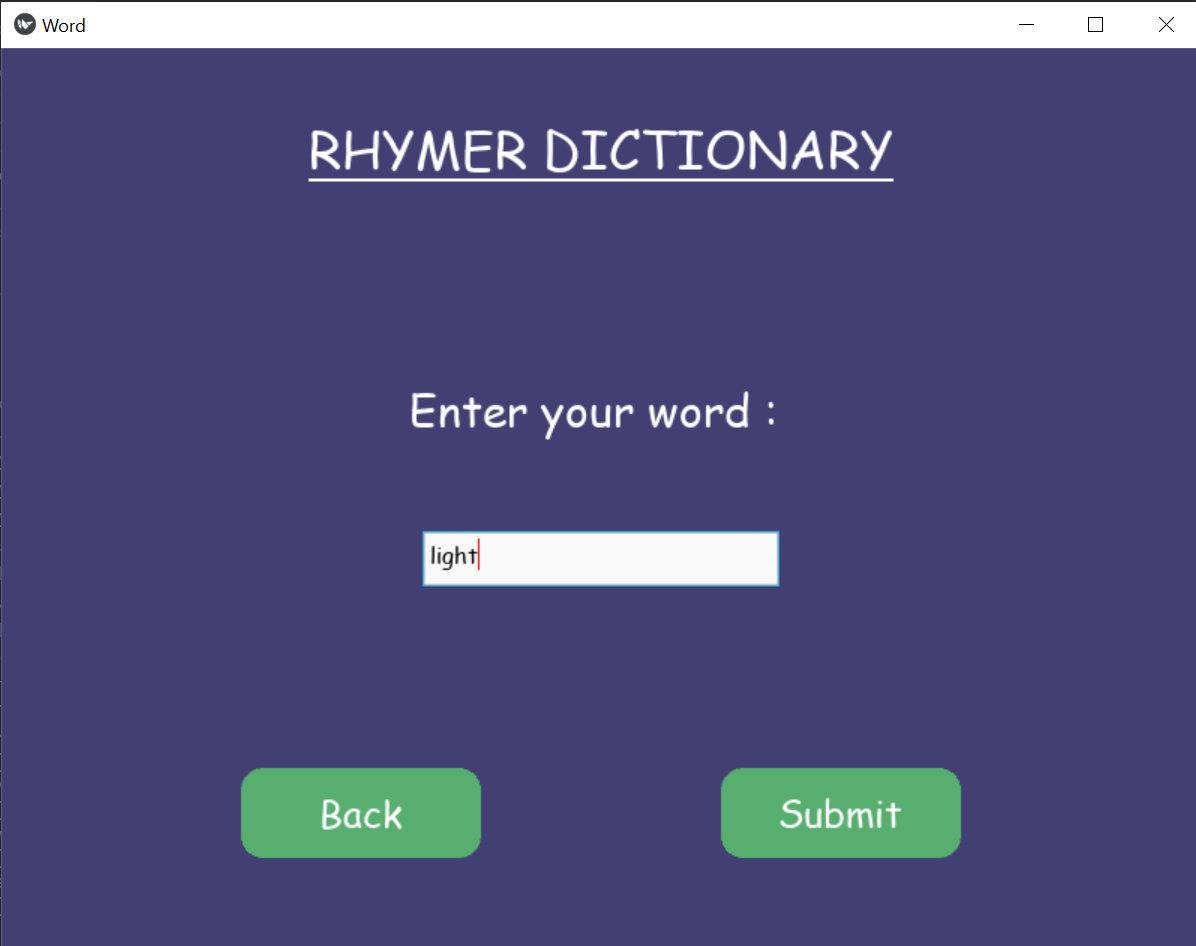
        if(new==""):

            return "No word found with specified meaning !"

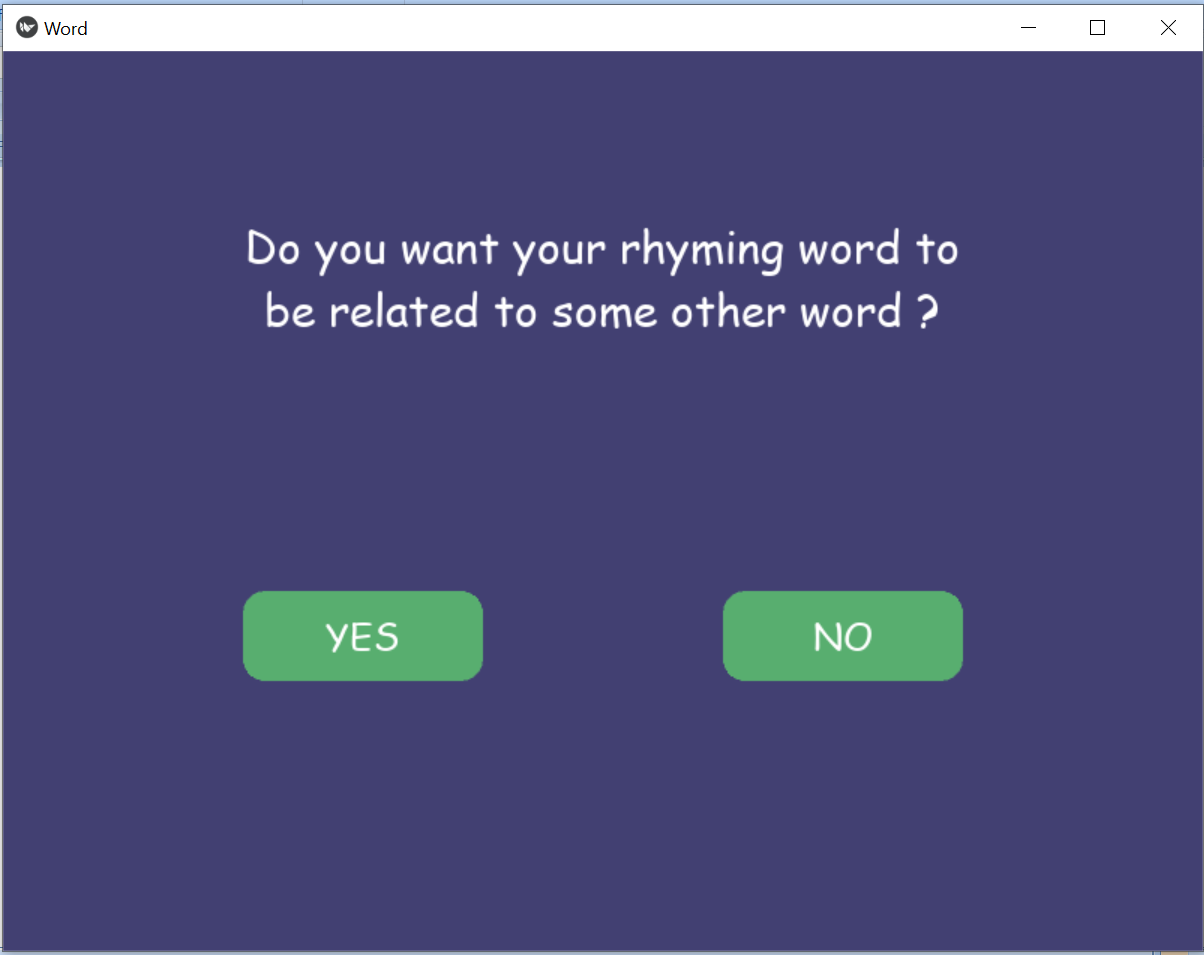
        return new

The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

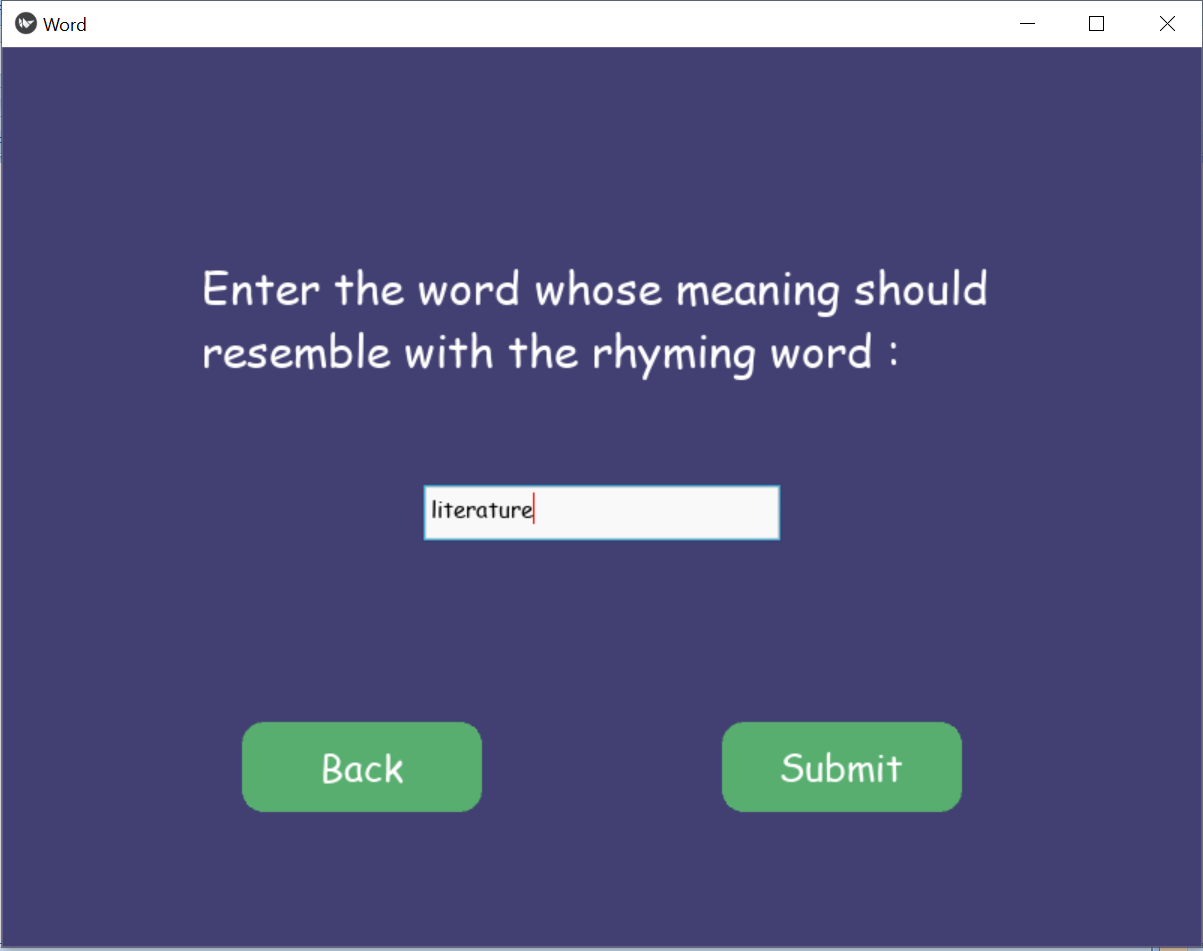
1. Taking Input:



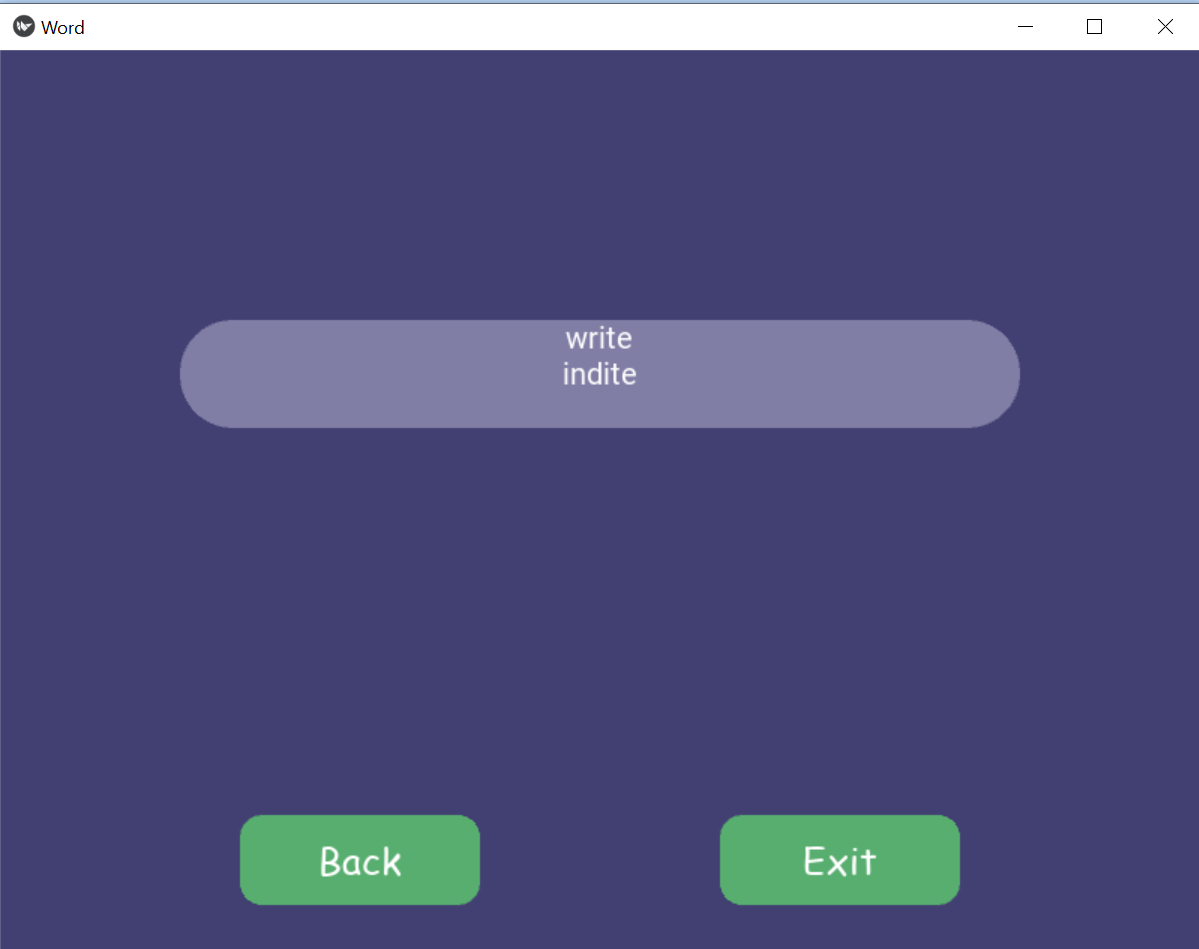
1. Choosing if the word should be of particular meaning:



1. Entering the word, the results should be related to:



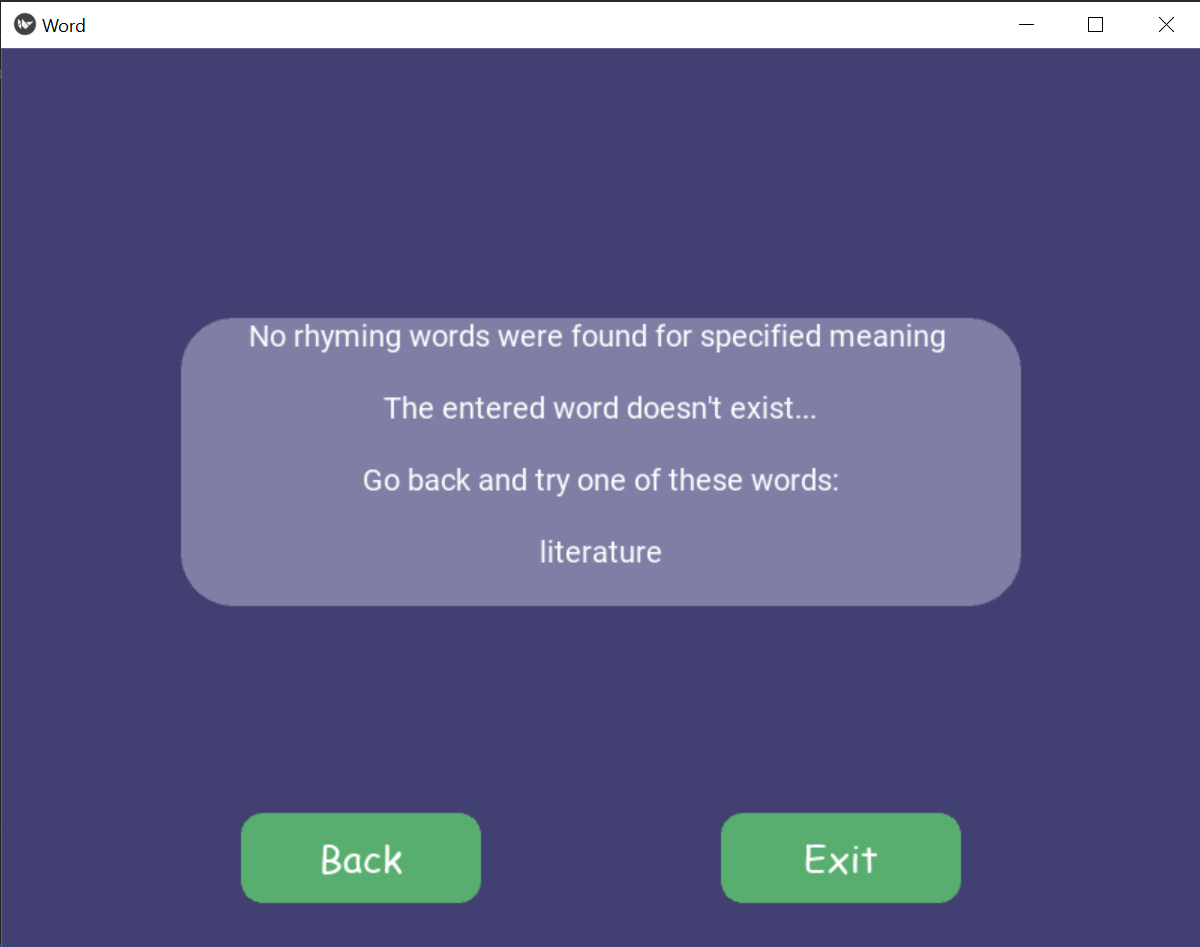
1. The results:



Hence the results comprise of the words that are related to literature and rhymes with Light.

If there are no words found, the application checks if there was spelling mistake and returns list of probable words.

For example if the meaning word is misspelled:



5.1.2 CROSSWORD HELPER

This section allows the user to find words for their crossword puzzle.

The user can enter the description they are looking for and they can define the length as well. The results are shown with an option to narrow them down by mentioning a fixed character and its position.

Logic Code with explanation in comments:

    def cross(self,word,le):

        if(word==""):

            return "No word entered !"

        if(le.isdigit()):

            leng = int(le)

        else:

            return "Length wasn't entered !"

        rr=[]

        #Splitting the words and putting them in the API's format

        words=word.split()

        w=""

        for a in words:

            if(w==""):

                w+=a

            else:

                w+="+"+a

        fn=[]

        res= requests.get("https://api.datamuse.com/words?ml="+w)

        for a in res.json():

            fn.append(a["word"])

        #The words related to description are stored in fn now.

        #Finding words with defined length

        new =""

        for l in fn:

            if len(l) == leng:

                new+=l+"\n"

        #returning error messages if no words are found.

        if(new==""):

            str="No words were found for the given inputs\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

                    str+=s+"\n"

            return str

        else:

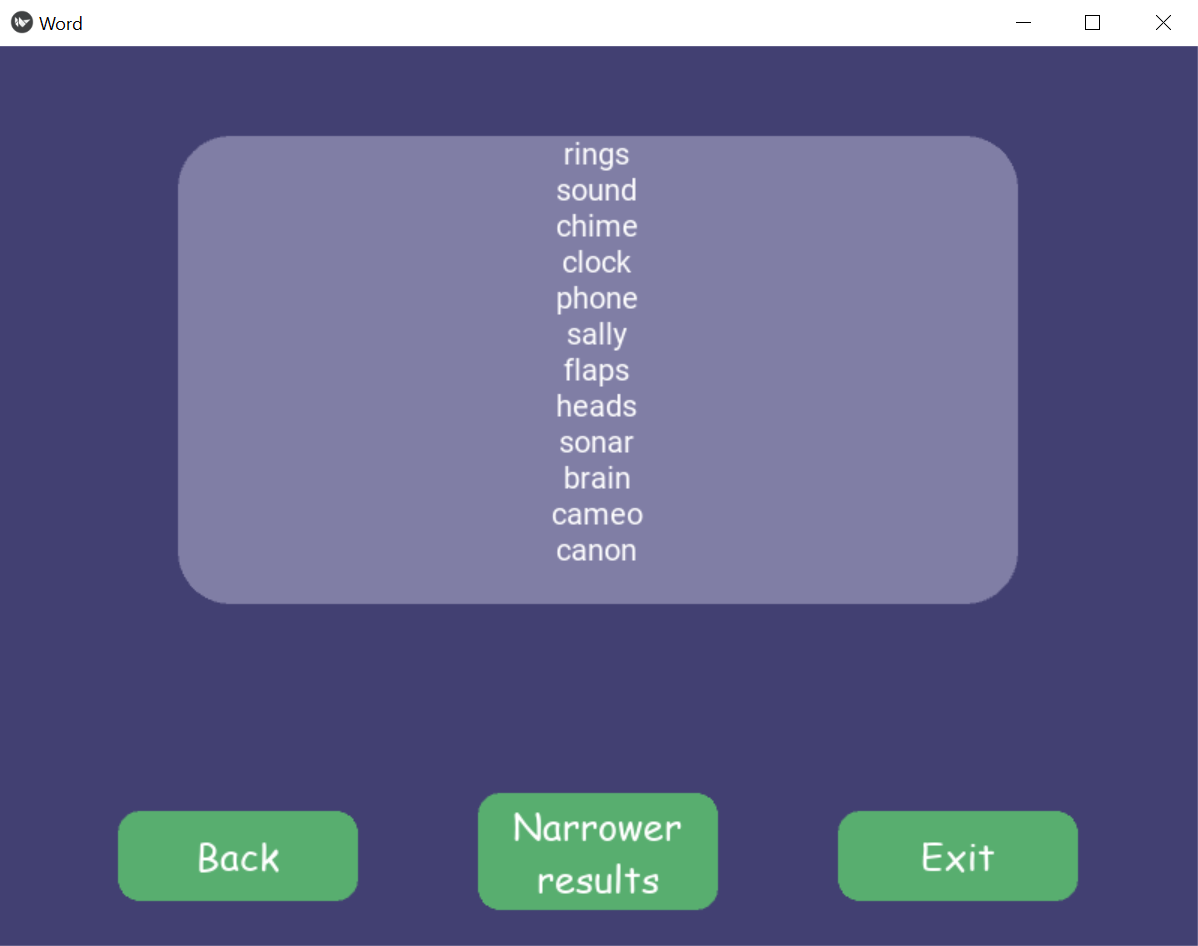
            return new

The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

1. Taking Input:



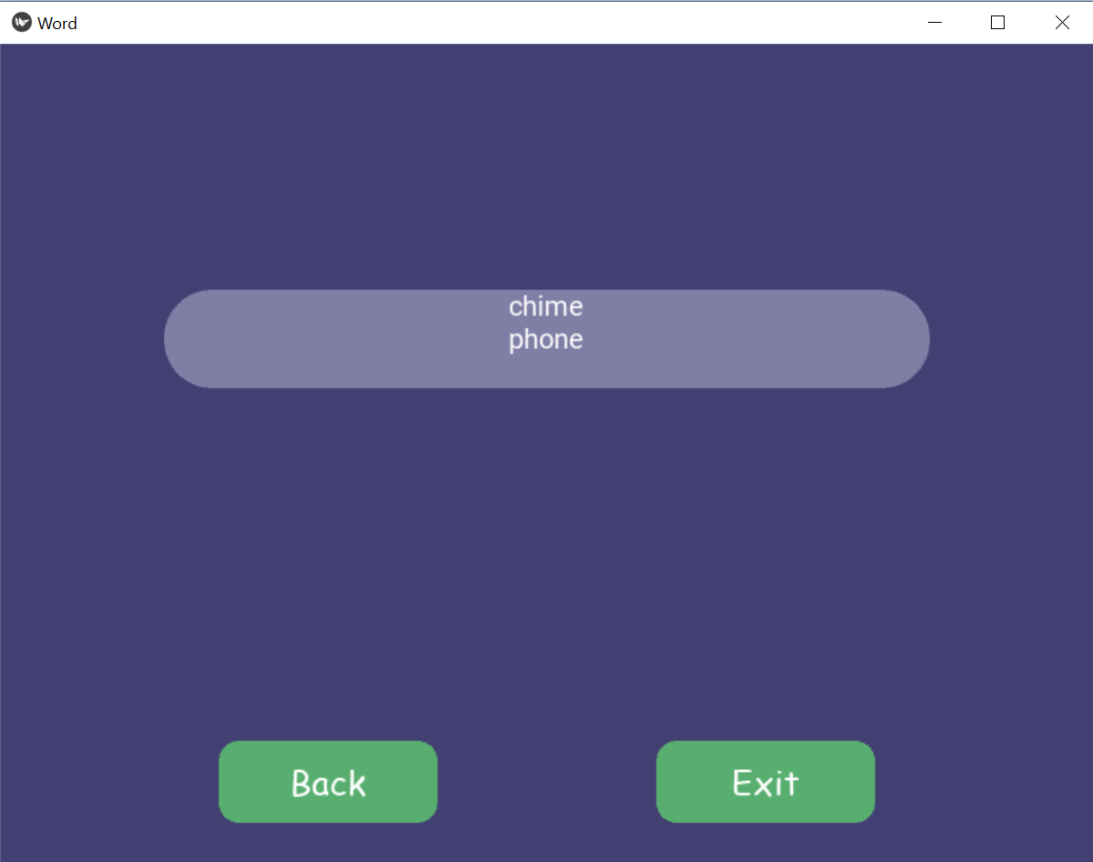
1. Results :



1. Narrower Results:



1. Narrow Results:



Code for Narrow Results:

def narrow(self,s,ch,pos):

        if(s==""):

            return "No word entered"

        posi= int(pos)-1

        words=s.splitlines( )

        new = ""

        for w in words:

            # print(w[posi]+" ")

            if w[posi] == ch+"" :

                new+=w+"\n"

        # print(new)

        if new =="":

            str="No words were found for the given inputs\n\n"

            if(spell.correction(s)!=s):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(s))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(s):

                    str+=s+"\n"

            return str

        return new

5.1.3 SCRABBLE HELPER

This is the section where user can find words for their scrabble game.

The user can enter the word fragment they want the resultant words to have, its position and the length of the whole word.

Logic Code with explanation in comments:

def scrabble(self,w,p,l):

        if(w==""):

            return "No word entered"

        with open('words\_alpha.txt') as word\_file:

            english\_words = set(word\_file.read().split())

        if(w==""):

            return "No Input"

        #Generating regex expression for all the cases:

        if(p=="^"):

            st="^"+w

            if(l.isdigit()):

                le=int(l)-len(w)

                if(le>0):

                    st = st+".{"+str(le)+"}"

                else:

                    return "Word Length can't be lesser than fragment length."

        elif (p=="$"):

            st=w+"$"

            if(l.isdigit()):

                le=int(l)-len(w)

                if(le>0):

                    st = ".{"+str(le)+"}"+st

                else:

                    return "Word Length can't be lesser than fragment length."

        elif(p==""):

            if(l.isdigit()):

                le=int(l)-len(w)

                if(le>0):

                    st = ".\*"+w+".\*"

                else:

                    return "Word Length can't be lesser than fragment length."

        elif(p.isdigit()):

            pp = int(p)-1

            st = ".{"+str(pp)+"}"+w

            if(l.isdigit()):

                le=int(l)-len(w) - pp

                if(le>0):

                    st = st+".{"+str(le)+"}"

        #matching and finding the words that contain the specified fragment in the specified position

        reg = re.compile(st)

        match=list(filter(reg.match,english\_words))

        new =""

        for word in match:

            if(l.isdigit()):

                if(len(word)==int(l)):

                    new+=word+"\n"

            else:

                new+=word+"\n"

        if(new==""):

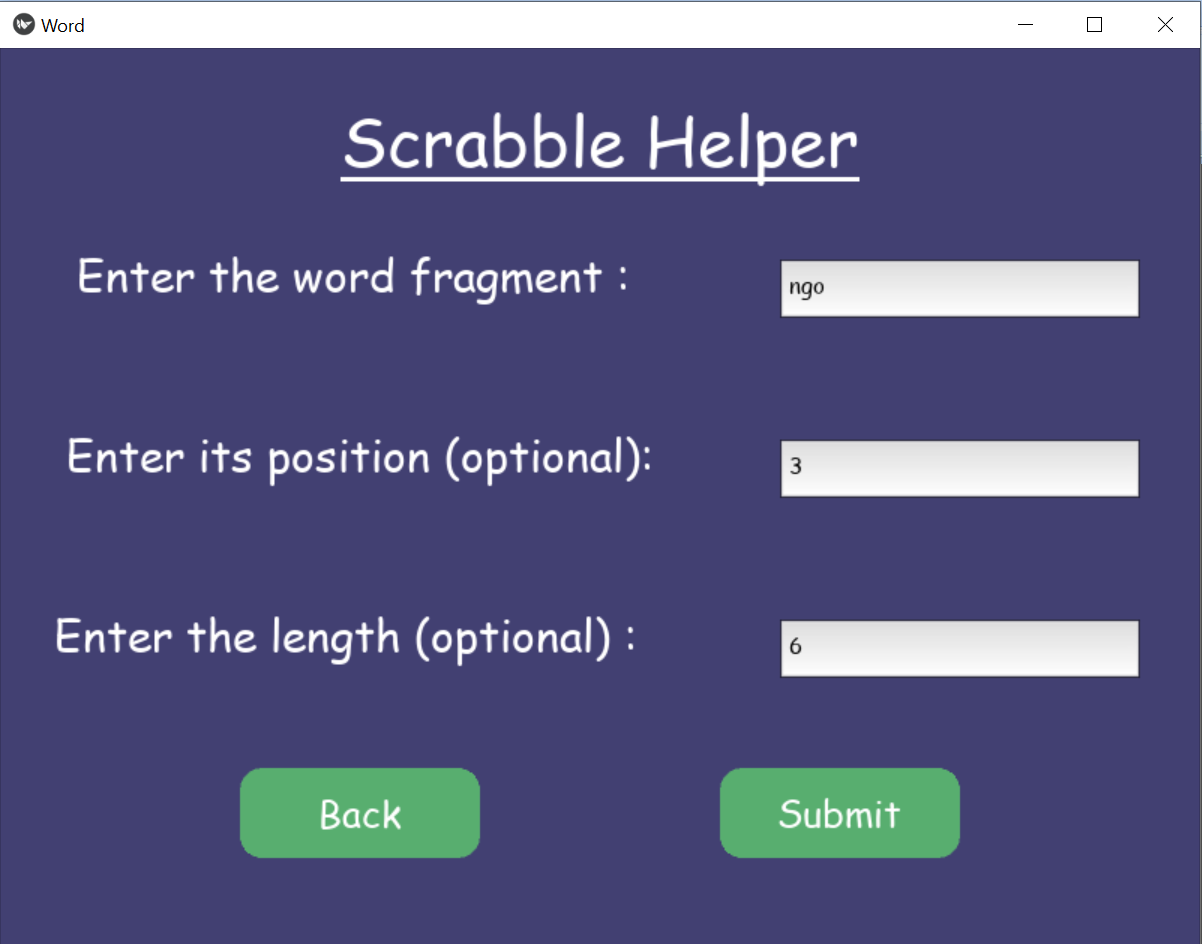
            return "No words were found, try entering different parameters."

        else:

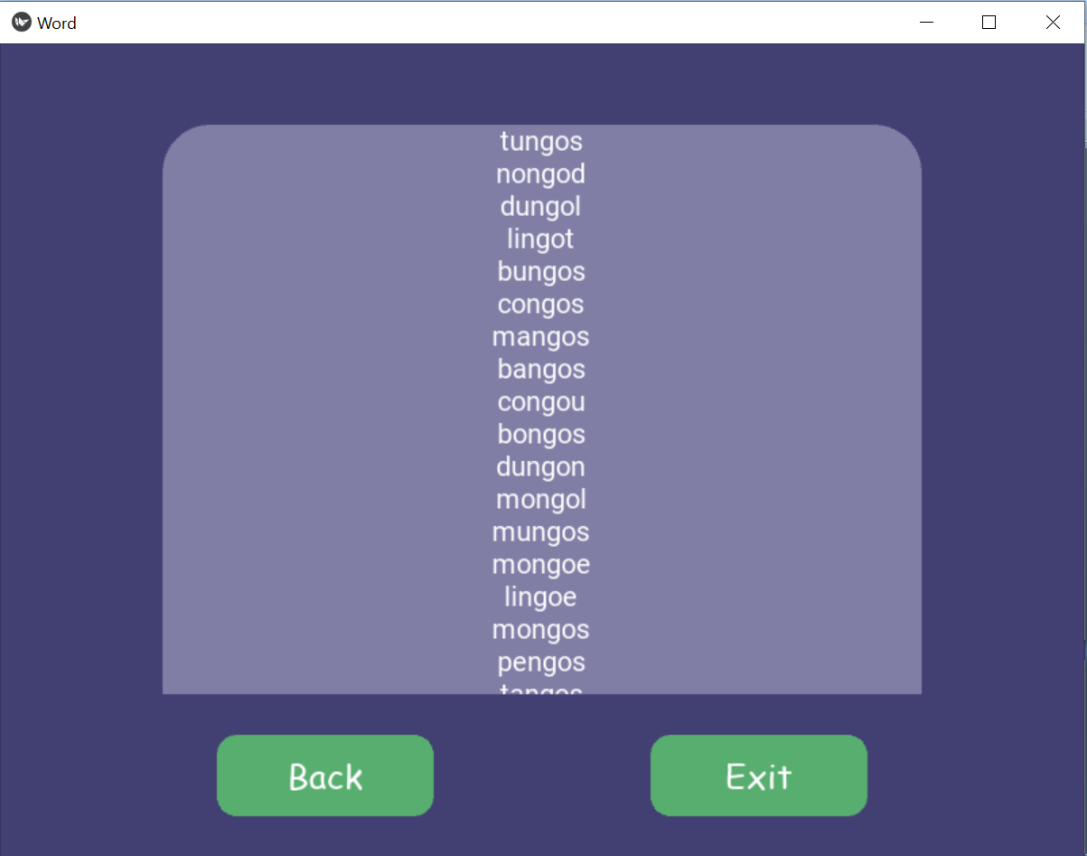
            return new

The code above is incorporated in a GUI by using Kivy. Below are the snippets with the workflow:

1. Taking Input:



1. Results:



Hence the user gets the list of words with length as 6 and has “ngo” at third position, thereby aiding them to get variety of words for their scrabble game.

The application makes use of a file which contains around 370150 English words.

It applies regular Expression to get the words with defined fragments at defined position of specifies length.

5.2 UNDERSTAND ZONE

Understand Zone is the section which allows user to understand a word that they enter.

The result is fetched from Oxford Dictionary’s official API which is free for a limited number of hits.

Below is the code:

def mean(self,word):

        if(word==""):

            return "No word entered"

        str=""

        response= requests.get("https://od-api.oxforddictionaries.com/api/v2/entries/en-gb/"+word+"?strictMatch=false",headers={"Accept": "application/json","app\_id": "c1498ba3","app\_key": "ec959282e97d787344cbe7cfeb13c965"})

        try:

            for a in response.json()["results"]:

                for b in a["lexicalEntries"][0]["entries"][0]["senses"]:

                    str+="Definition:\n "+b["definitions"][0]+"\n"

                    str+="Example: \n"+b["examples"][0]["text"]+"\n\n"

        except KeyError:

            str="No meanings were found for the given inputs\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

                    str+=s+"\n"

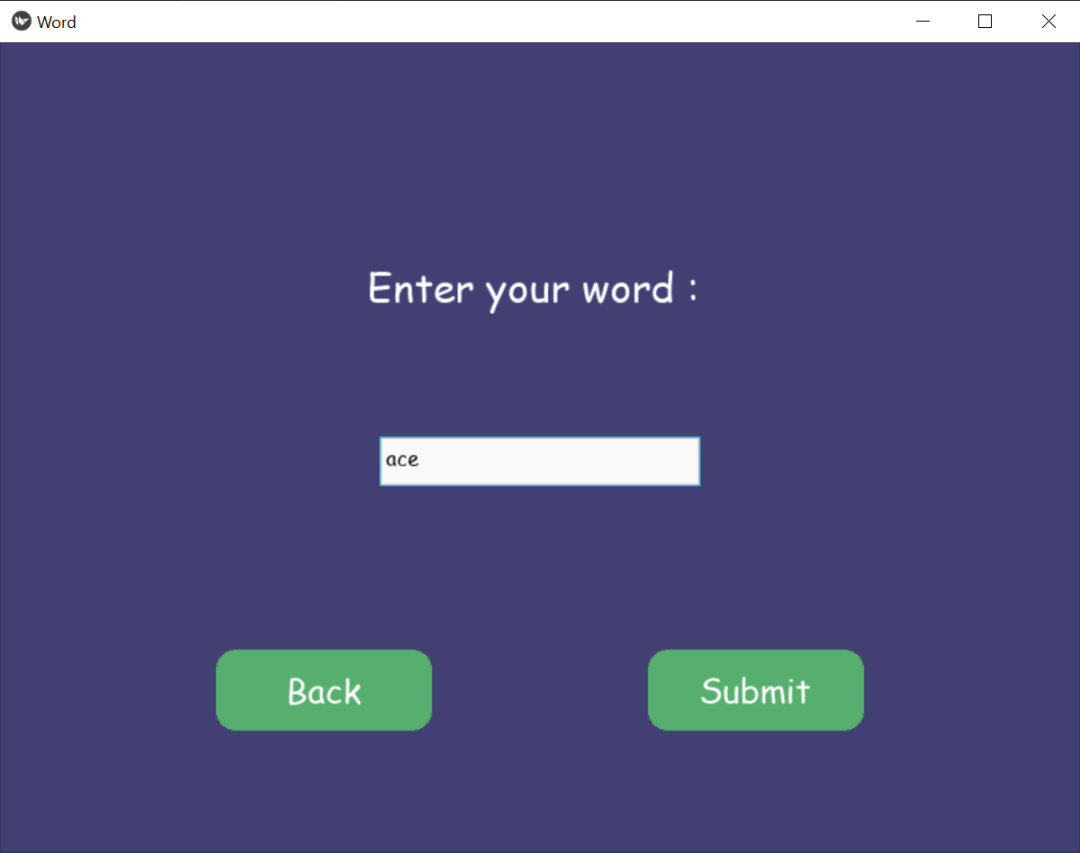
        return str

The result comprises of multiple meanings along with supporting meanings.

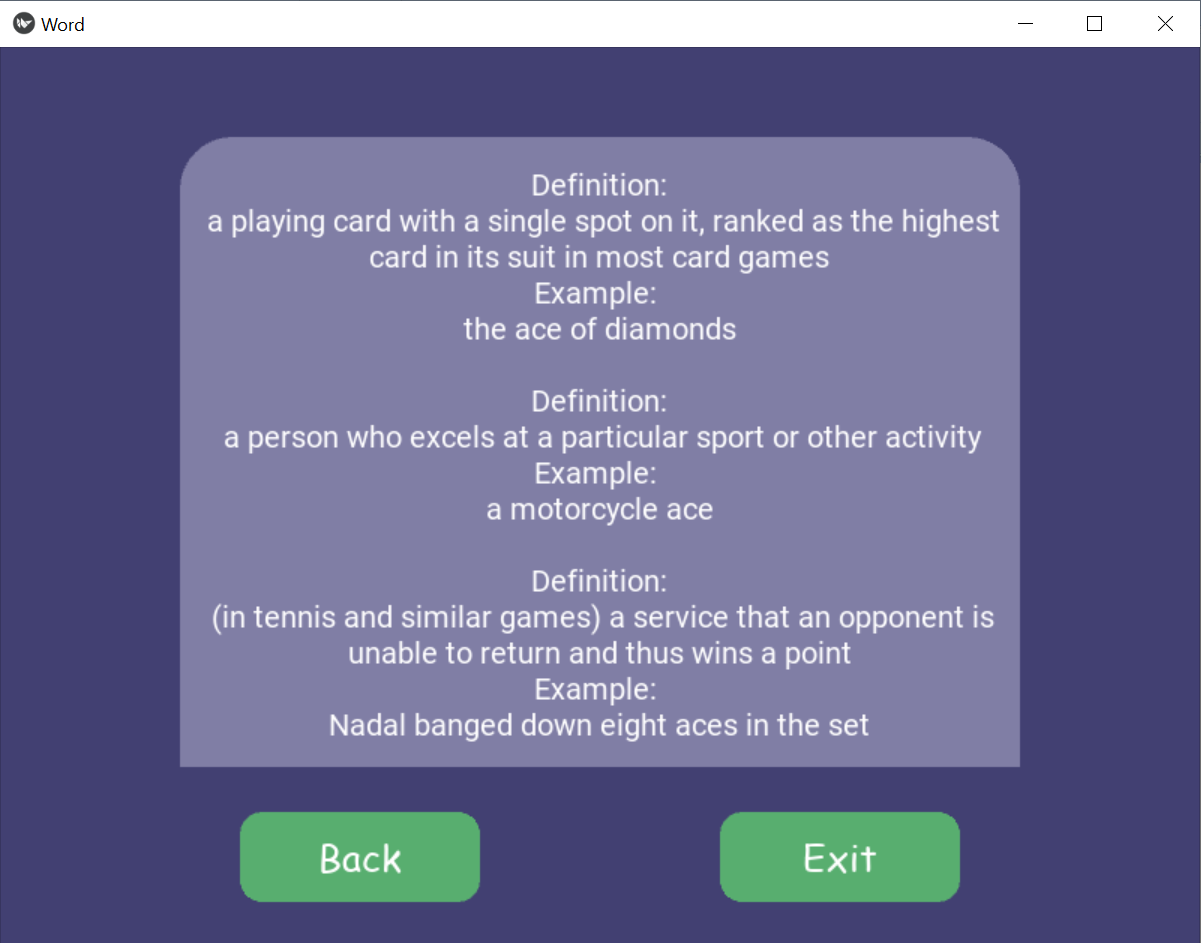
If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the spellcheck library.

Snippets and workflows:

1. Input:



1. Output:

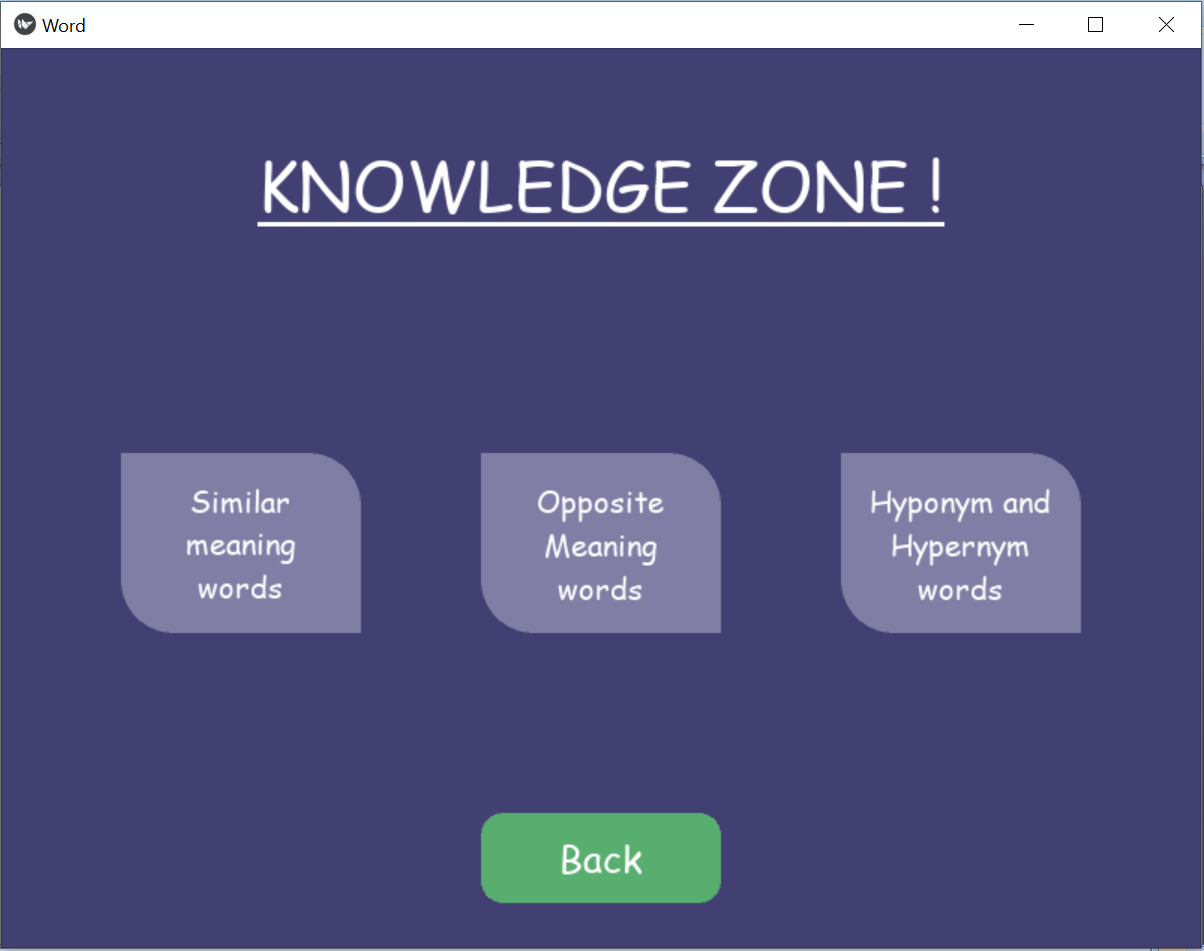


5.3 KNOWLEDGE ZONE

Knowledge Zone is the section in which user can know more related words to a given word.

There are three sections inside Knowledge Zone where they can find synonyms, antonyms, hyponym, hypernym and holonyms.

All these sections make use of Wordnet which stores a lot of words related to a given word.



5.3.1 SIMILAR MEANING WORDS

User can find similar meaning words, i.e. synonyms of a word by entering a word in this section.

Code snippet:

    def syn(self,word):

        synonyms = []

        if(word==""):

            return "No word entered"

        for syn in wordnet.synsets(word):

            for l in syn.lemmas():

                synonyms.append(l.name())

        if(len(synonyms)>0):

            ss=set(synonyms)

            new = ""

            for x in ss:

                new += x+"\n"

            return new

        else:

            str="No synonyms were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

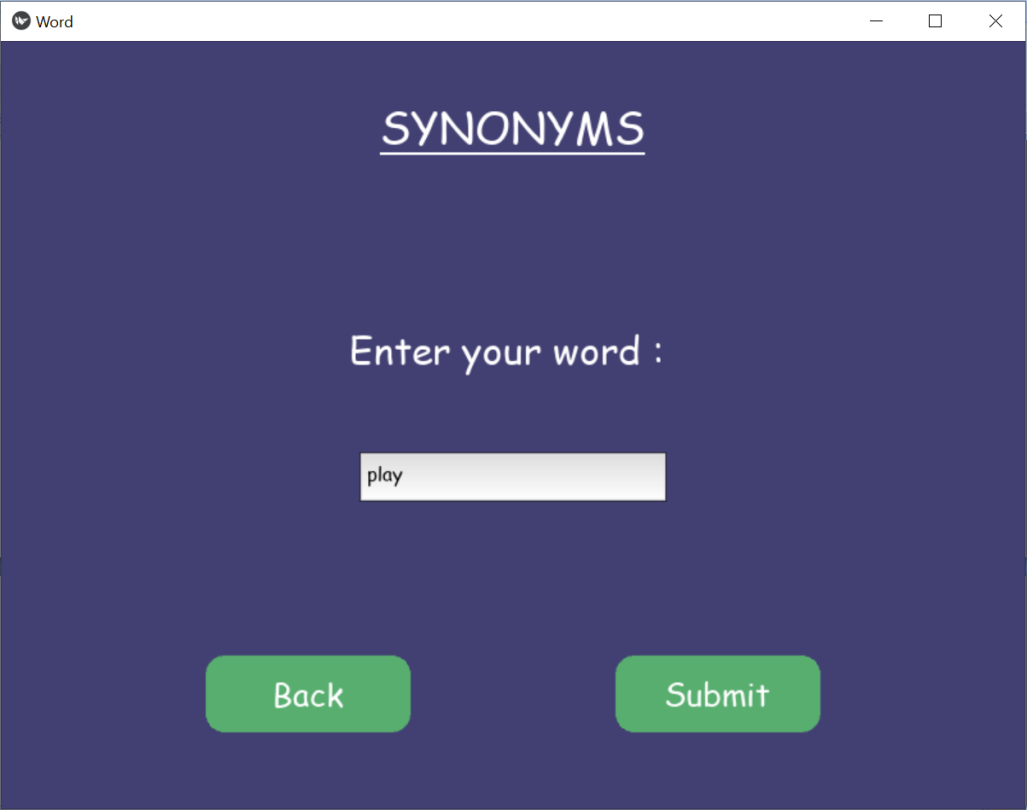
                    str+=s+"\n"

            return str

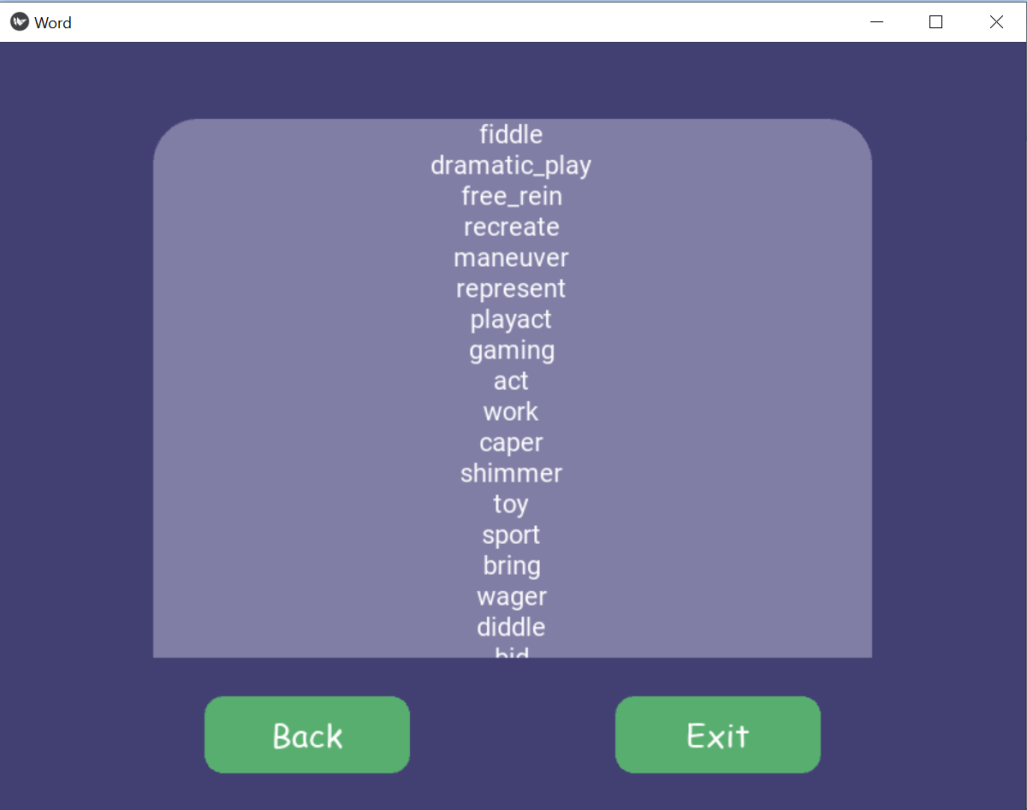
If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the “spellcheck” library.

Snapshots and workflow:

1. Input:



1. Output:



5.3.2 OPPOSITE MEANING WORDS

In this section, user can find opposite meaning words, i.e. Antonyms by entering a word.

This also makes use of wordnet’s synsets to find antonyms.

Code Snippet:

def ant(self,word):

        antonyms = []

        if(word==""):

            return "No word entered"

        for syn in wordnet.synsets(word):

            for l in syn.lemmas():

                if l.antonyms():

                    antonyms.append(l.antonyms()[0].name())

        if(len(antonyms)>0):

            ss=set(antonyms)

            new = ""

            for x in ss:

                new += x+"\n"

            return new

        else:

            str="No antonymns were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

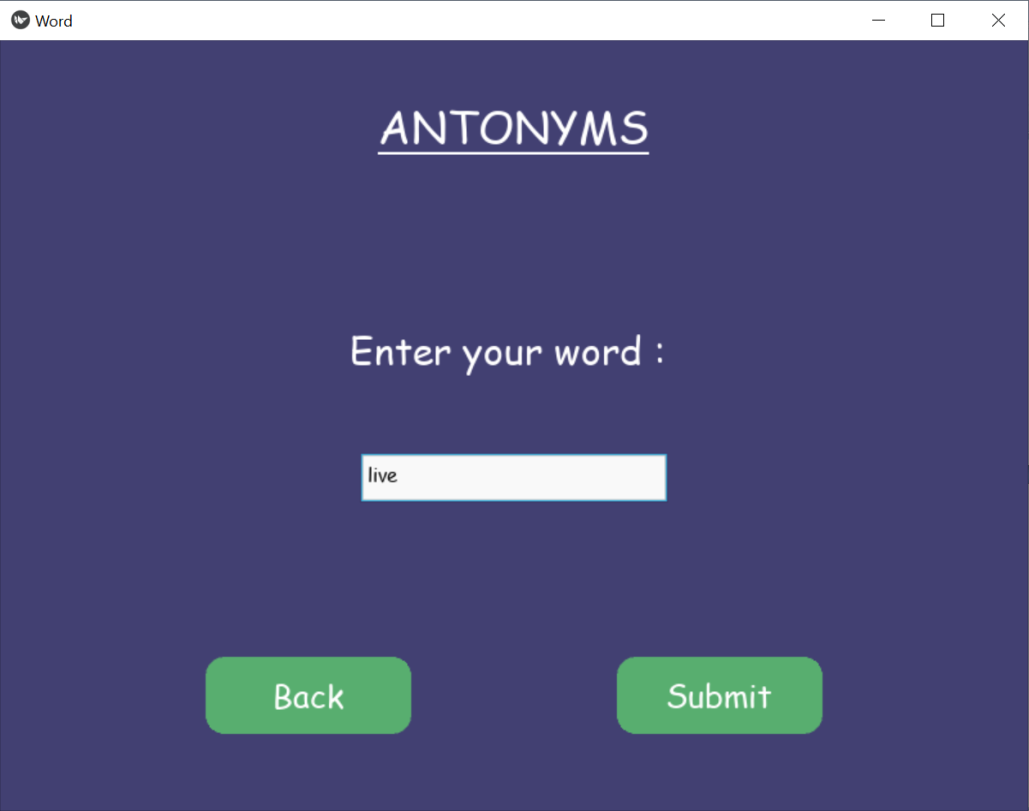
                    str+=s+"\n"

            return str

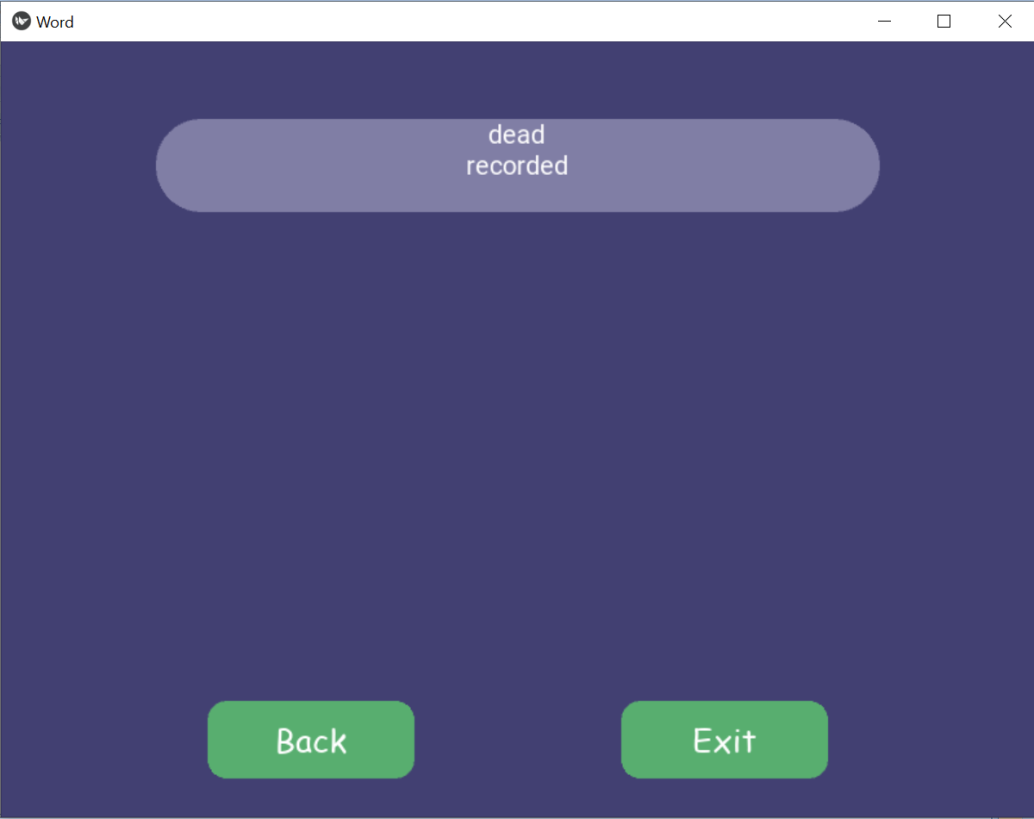
If the user enters a wrongly spelled word, he is prompted with nearest correctly spelled words with the help of the “spellcheck” library.

Snapshots and workflow:

1. Input:



1. Output:



5.3.3 HYPERNYM – HYPONYM - HOLONYM

This section allows user to explore hypernyms, hyponyms and holonyms of entered words.

Hypernym are the broader category term for a given word, for example ‘Red’, ‘Purple’, ‘Green’ all have a common hypernym as ‘Color’.

Hyponym are the narrow category term for a given word for example ‘Red’ is a hyponym of ‘Color’.

Holonym defines the relationship between a term denoting the whole and a term denoting a part of, or a member of, the whole. For example ‘Tree’ is a holonym of ‘Trunk’.

Code Snippet:

Hypernym:

def hyper(self,word):

        sn=""

        if(word==""):

            return "No word entered"

        syns=wordnet.synsets(word)

        for syn in syns:

            s=syn.hypernyms()

            for q in s:

                sn+=q.lemma\_names()[0]+"\n"

        if(sn!=""):

            return sn

        else:

            str="No words were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

                    str+=s+"\n"

            return str

Hyponym:

def hypo(self,word):

        sn=""

        if(word==""):

            return "No word entered"

        syns=wordnet.synsets(word)

        for syn in syns:

            s=syn.hyponyms()

            for q in s:

                sn+=q.lemma\_names()[0]+"\n"

        if(sn!=""):

            return sn

        else:

            str="No words were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

                for s in spell.candidates(word):

                    str+=s+"\n"

            return str

Holonym:

def holo(self,word):

        sn=""

        if(word==""):

            return "No word entered"

        syns=wordnet.synsets(word)

        for syn in syns:

            s=syn.member\_holonyms

            for q in s:

                sn+=q.lemma\_names()[0]+"\n"

        if(sn!=""):

            return sn

        else:

            str="No words were found for the given input\n\n"

            if(spell.correction(word)!=word):

                str+="The entered word doesn't exist...\n\n"

                if len(spell.candidates(word))>0 :

                    str+="Go back and try one of these words:\n\n"

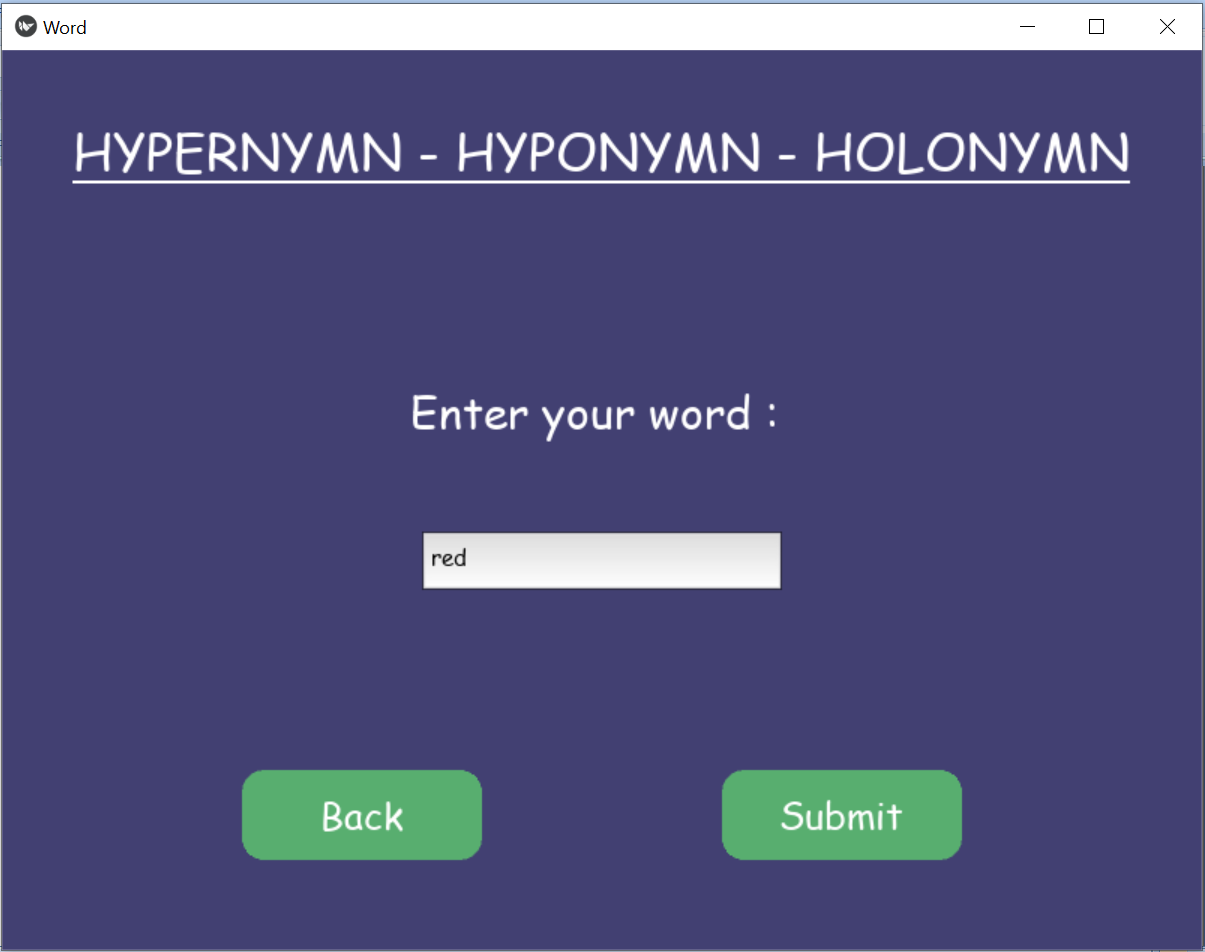
                for s in spell.candidates(word):

                    str+=s+"\n"

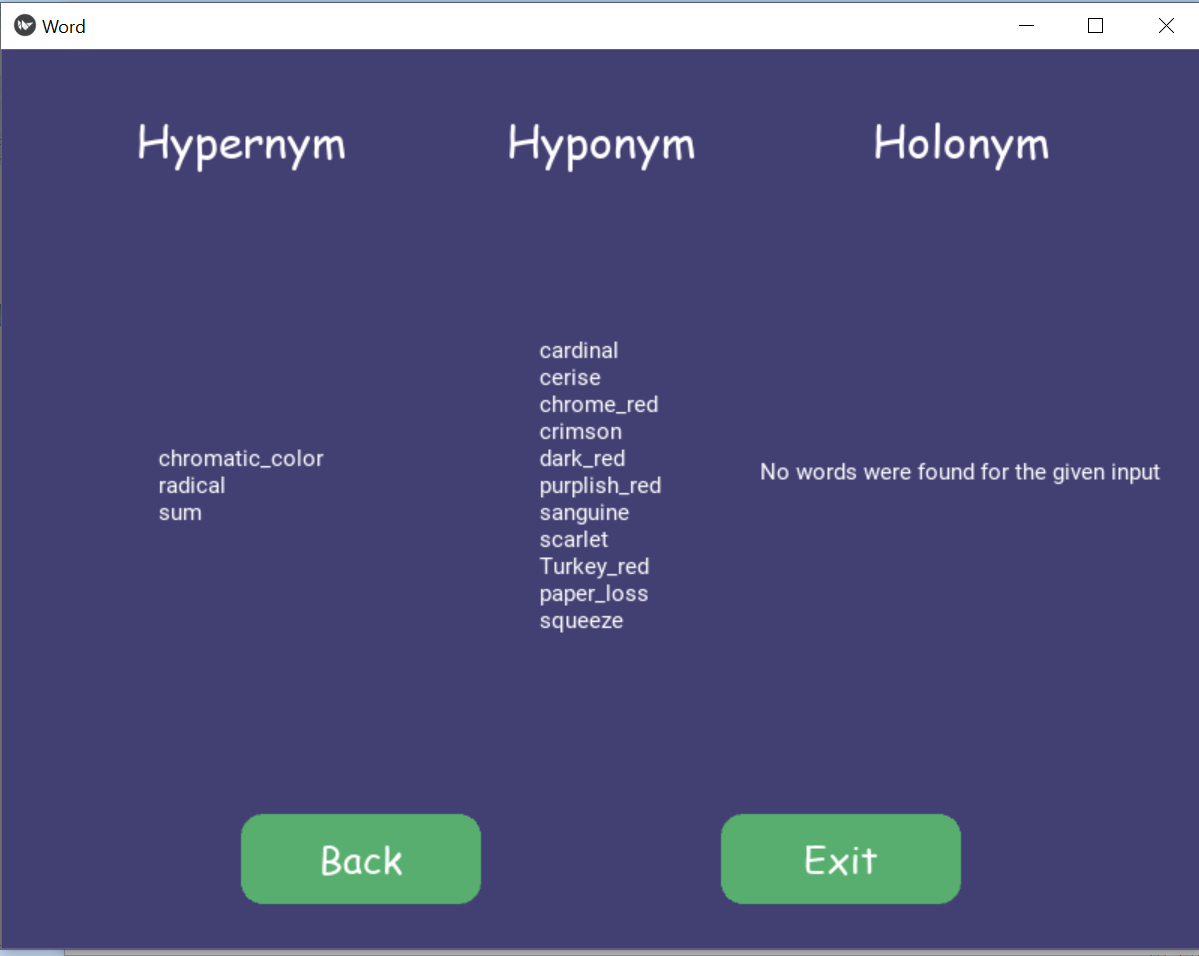
            return str

Snapshots:

1. Input:

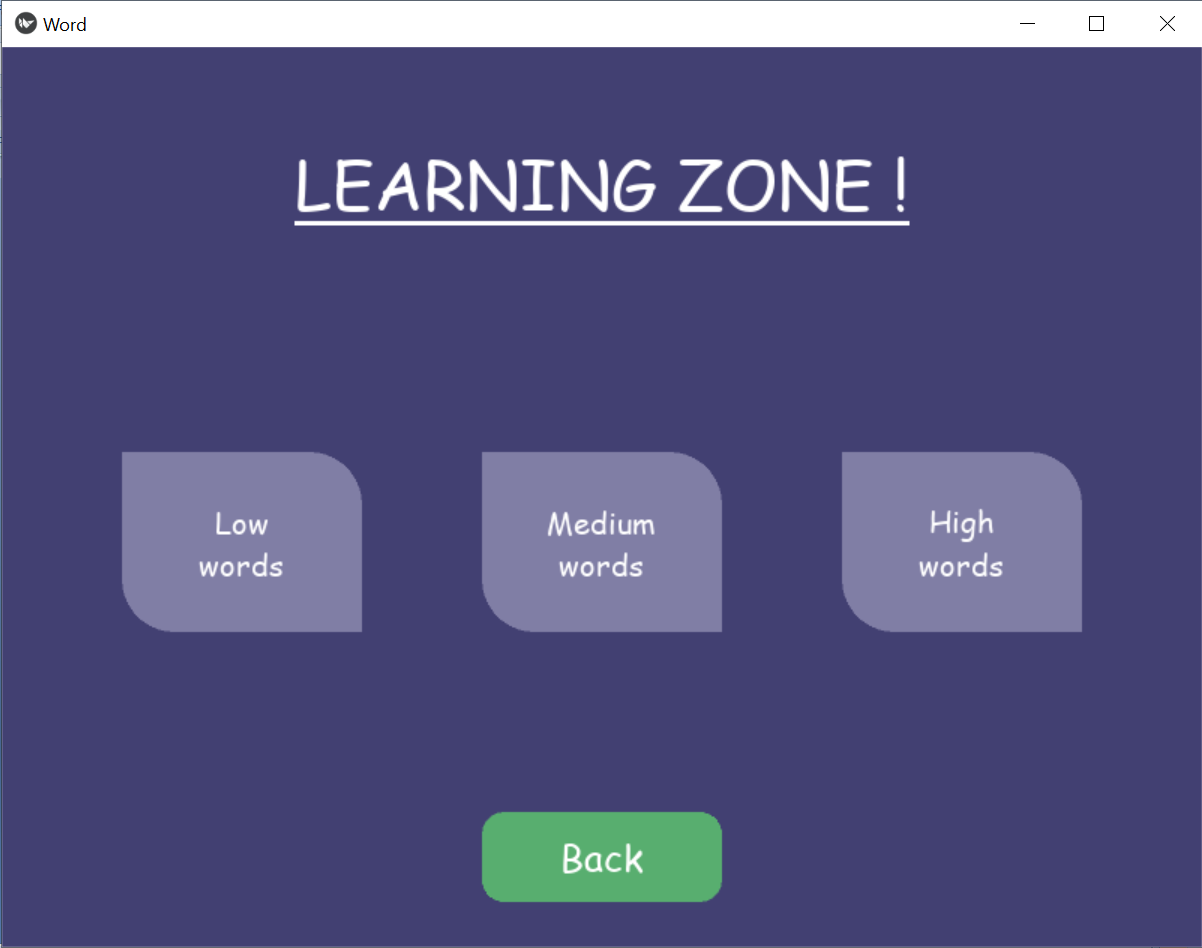


1. Output:

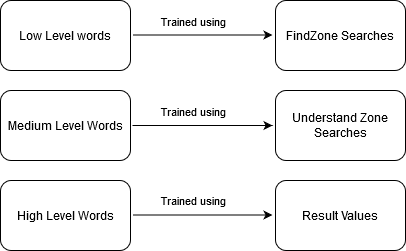


5.4 LEARNING ZONE

Learning Zone is the section which has three sections which shows low, mid and high level words based on user’s usage.



The levels are trained with respective user’s usages:



5.4.1 LOW LEVEL WORDS

Low level words are the easy words, Findzone searches are done using words which the user already knows, hence these words can be used to show low level words.

5.4.2 MEDIUM LEVEL WORDS

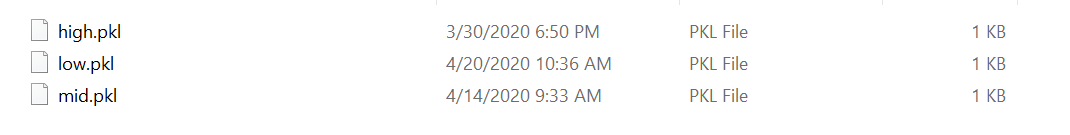
Medium Level words are the words which the user genuinely wants to know, hence understand zone searches (where user enters the words that they want to understand) are used to train medium level model.

5.4.3 HIGH LEVEL WORDS

High level words are the words that are very less frequently used and are trained by the less frequently used words from results.

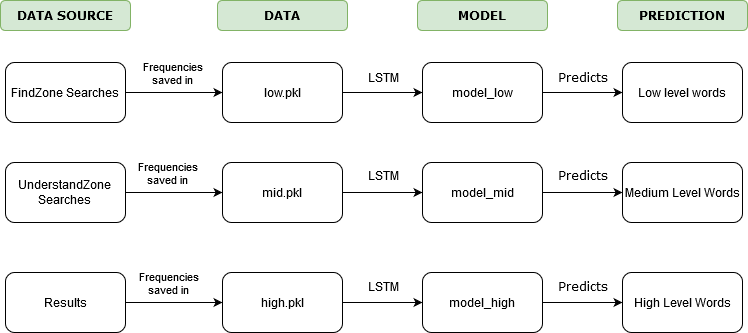
5.4.4 TRAINING and PREDICTION:

Training is done on the data stored in either of the three files:



These files contains the frequency of the words according to the mapping shown in previous section.

The toughness of a word is determined by how frequently it is being used in English language. It is determined using the library WordFreq. Hence frequencies of words from respective modules are stored in their respective pickle files which are then used to train respective models.



Generic code for training:

def train(data\_path):

  with open("./data/"+data\_path+".pkl", 'rb') as f:

    data\_list = pickle.load(f)

  df = pd.DataFrame(data\_list, columns=['frequency'])

  frequency\_train = df['frequency'].values.reshape((-1, 1))

  look\_back = 5

  train\_generator = TimeseriesGenerator(frequency\_train, frequency\_train, length=look\_back, batch\_size=20)

  from keras.models import Sequential

  from keras.layers import LSTM, Dense

  model = Sequential()

  model.add(

      LSTM(10,

          activation='relu',

          input\_shape=(look\_back,1))

  )

  model.add(Dense(1))

  model.compile(optimizer='adam', loss='mse')

  num\_epochs = 25

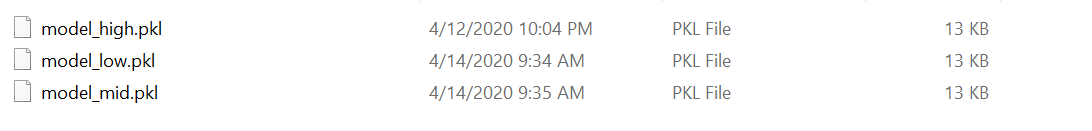
  # train\_generator = train\_generator.reshape(-1, -1, 1)

  model.fit\_generator(train\_generator, epochs=num\_epochs, verbose=1)

  with open('./models/model\_' + data\_path +".pkl" ,'wb') as f:

    pickle.dump(model, f)

The trained models can be retrained and are kept separately in folder.



These models are then used to predict words using the predict function :

def predict(mtype, num\_prediction=1):

  with open("./data/"+mtype+".pkl", 'rb') as f:

    data\_list = pickle.load(f)

  model=None

  look\_back=5

  with open('./models/model\_' +mtype+".pkl", 'rb') as f:

    model=pickle.load(f)

  df = pd.DataFrame(data\_list, columns=['frequency'])

  frequency = df['frequency'].values[-look\_back:]

  prediction\_list = frequency[-look\_back:]

  for \_ in range(num\_prediction):

      x = prediction\_list[-look\_back:]

      x = x.reshape((1, look\_back, 1))

      out = model.predict(x)[0][0]

      prediction\_list = np.append(prediction\_list, out)

  prediction\_list = prediction\_list[look\_back-1:]

  return prediction\_list

5.5 CONCLUSION:

In conclusion, WORDZONE is a platform for people to look up a lot of word related searches whether it is Finding a word, Understanding a word, Knowing a related word or Learning a new word, WORDZONE covers all.

5.6 FUTURE WORK

**References:**

[1].Liu Vinci &Curran James(2006),Web Text Corpus for Natural Language Processing.

[2] Loper, Edward & Bird, Steven. (2002). NLTK: the Natural Language Toolkit. CoRR. cs.CL/0205028. 10.3115/1118108.1118117.

[3]Miller, George & Beckwith, R. &Fellbaum, Christiane & Gross, Derek & Miller, Katherine. (1991). Introduction to WordNet: An On-line Lexical Database\*. 3. 10.1093/ijl/3.4.235.

[4]Rajani S, M. Hanumanthappa, 2016, “Techniques of Semantic Analysis for Natural Language Processing – A Detailed Survey”

[5] Dr. M Hanumanthappa, Rashmi S, Jyothi N M, “Impact of Phonetics in Natural Language Processing: A Literature Survey”, IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.

[6]PrimozPodrzaj . A brief demonstration of some Python GUI libraries Proceedings of The 8th International Conference on Informatics and Applications ICIA2019, Japan, 2019

# [7] Hassanin M. Al-Barhamtoshy, Fatimah M. Mujallid,“Building Mobile Dictionary” 2013

[8]Byrd, Roy &Chodorow, Martin. (2002). Using An On-Line Dictionary To Find Rhyming Words And Pronunciations For Unknown Words. 10.3115/981210.981244.

[9] Time Series forcasting - Siami Namini, Sima & Tavakoli, Neda & Siami Namin, Akbar. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. 1394-1401. 10.1109/ICMLA.2018.00227.

[10] Robyn Speer. Wordfreq is a Python library for looking up the frequencies of words in many languages, based on many sources of data.

[11] Python Library for object serialization, comparison with marshal and json available [here](https://docs.python.org/3/library/pickle.html).