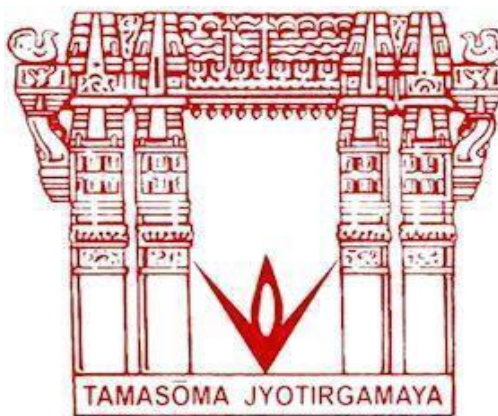


# **INTERACTIVE DICTIONARY**

A Course Based Project Submitted in Partial Fulfilment of the

Requirements for the Award of the degree of

## **BACHELOR OF TECHNOLOGY COMPUTER SCIENCE AND ENGINEERING**



Under the Guidance of  
Mrs.E.Lalitha  
(Assistant Professor,VNRVJIET)

Submitted by:

21071A6207– B. Varsha Reddy

21071A6211 – Ch.Keerthana

# CERTIFICATE

This is to Certify that B. Varsha Reddy (21071A6207),  
Ch. Keerthana (21071A6211) of CSE-CYS, DS  
&(AI&DS) Department of VNRVJIET have successfully  
completed their project work entitled “Bookshop  
Inventory System” in partial fulfilment of the  
requirements for the award of the Bachelor of Technology  
degree during the Academic year 2022-2023.

Project Guide Mrs.E.Lalitha

Assistant prof.& Internal guide

Dept. of CSE-CYS, DS &(AI&DS) VNRVJIET

Head of Department Dr. M. RAJA SEKHAR

Prof. and Head

Dept. of CSE-CYS, DS&(AI&DS)

VNRVJIET

# ACKNOWLEDGEMENT

Behind every achievement lies the heartfelt gratitude to those who activated in completing the project. To them we lay the words of gratitude within us.

We are indebted to our venerable principal Dr. C.D. NAIDU for this inflicting devotion, which lead us to complete this project. The support, encouragement given by him, and his motivation led us to complete the project.

We express our sincere thanks to internal guide Mr.E.Lalitha and Head of the Department Dr. M. RAJA SHEKHAR for having provided us with a lot of facilities to undertake the project work and guide us to complete the project.

We take the opportunity to express thanks to our faculty of the Dept. of COMPUTER SCIENCE AND ENGINEERING-CYS and remaining members of our college VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY who extended their valuable support in helping us to complete the project in time.

B.Varsha Reddy

(21071A6207)

Ch.Keerthana

(21071A6211)

# ABSTARCT

## What is interactive dictionary?

A dictionary is a kind of data structure that stores items in key-value pairs. A key is a unique identifier for an item, and a value is the data associated with that key. Dictionaries often store information such as words and definitions, but they can be used for much more. Dictionaries are mutable in Python, which means they can be changed after they are created. They are also unordered, indicating the items in a dictionary are not stored in any particular order.

It is always tedious work to find the meaning of a word in the dictionary, isn't it? But what if you create your dictionary which can find you the meaning of the word just in a few seconds? You can build an interactive dictionary using python language by just using basic concepts like JSON, functions, and conditional statements.

Create a dictionary in Python which the user to retrieve definitions for user, if user made a typo while entering the word, and if the word has more than one definition then retrieve them all.

Dictionary comprehension is a feature that allows you to create a dictionary from a list or another dictionary in a single line of code. It is a very concise way of creating dictionaries and can be used to perform various operations on dictionaries. The expression of Dictionary comprehension is (key: value) followed by a for statement within the curly braces { }

Dictionaries have several built-in functions that allow you to perform various operations on them. `all()`, `any()`, `len()`, `cmp()`, `sorted()`, etc. are the most common dictionaries functions.

In addition to its basic function of defining words, a dictionary may provide information about their pronunciation, grammatical forms and functions, etymologies, syntactic peculiarities, variant spellings, and antonyms.

Programming languages all come with a variety of data structures, each suited to specific kinds of jobs. Among the data structures built into Python, the dictionary, or Python dict, stands out. A Python dictionary is a fast, versatile way to store and retrieve data by way of a name or even a more complex object type, rather than just an index number.

This particular article is going to be divided into two parts, for the first part we are going to develop a terminal based application and later we will move into developing an interactive GUI app.

Despite the steps taken, that is either GUI based or Terminal based our dictionary will work the same performing the same operations, that is:

The first step, the JSON file will be loaded into python. Get user input  
Cross check the meaning of the word from the JSON file

## **Creating Dictionary App:**

### Terminal SetUp

In order for us to create our interactive dictionary and use it in the terminal, we will need:

- A .py file – which will contain our code.  
A data file—which is a JSON file that contains vocabularies that will be checked against giving meaning of words asked for.
- difflib – help us compare various sequences and give us a list of words which are close to what the user intended to search for.

The first step will be for us importing the libraries that we will be using, and that is done as shown below.

```
import json
from difflib import
get_close_matches
```

Next we load up the JSON file.

```
data =
json.loads(open('data.json').re
ad())
```

At this point now we can go ahead and write our main function which will contain the code shown below.

We have defined a function: definition.

Converted all user input into lowercase to match words in JSON file.

If statement to check if words given exist, suggest possible words, and return an error if word does not exist.

```
def definition(name):  
    name = name.lower()  
    if name in data:  
        return data[name]  
    elif len(get_close_matches(name, data.keys())) > 0:  
        check = input("Did you mean %s instead? Enter Y if yes, otherwise N to  
exit: " %  
                        get_close_matches(name, data.keys())[0])  
        if check == "Y":  
            return data[get_close_matches(name, data.keys())[0]]  
        elif check == "N":  
            return "The word doesn't exist. Please double check it."  
        else:  
            return "We didn't understand your entry."  
    else:  
        return "Sorry, this word is not an English word. Please double check your  
spelling."
```

Next, we now pass in the method that will take in the user input and check it against the set of words passed in the JSON file.

```
word = input('Enter a name: ')
```

The last step is to add an output format. We need to get our answers arranged in an easier manner to read, and in order to achieve that we make use of the **if statement** and **for loop**. Which will list all meanings in a separate line in the terminal.

To achieve that, we make use of the code below.

```
output = definition(word)
```

```
if type(output) == list:
```

**for item in output:**

**print(item)**

**else:**

**print(output)**

Now let's go ahead and test out our program by running it on different instances. If you check an English word, for example the word "program" the output will be as shown below.

```
$ python find.py
Enter a name: program
A scheme of action, a method of proceeding, or a series of steps, thought out in advance to accomplish a
A software application, or a collection of software applications, designed to perform a specific task.
To enter a program or other instructions into a computer (or other electronic device) to instruct it to
To arrange the schedule of an event.
```

### **GUI based Dictionary :**

In this particular section, we will do things a little bit differently. Since we will need an interface we will interact with we are going to use different modules compared to our terminal app.

In this case apart from having a .py file, we are going to make use of:

- PyDictionary – A Python module that will help us get meanings, synonyms and antonyms of words.
- Tkinter – A python framework that will enable us to create GUI elements using its widgets found in the Tk toolkit.

### **Step 1: Installing the packages**

To install PyDictionary, we will use the pip command on either the terminal or command prompt.

`pip install PyDictionary`

Similarly to install

Tkinter

we make use of pip by executing the command below:  
`pip install tk`

## Step 2: Import the installed packages

In order for us to use the two modules that we have installed via the terminal, we will need to import them. The code below allows us to do this:

```
from tkinter import *  
from PyDictionary import PyDictionary
```

First, we import tkinter with all its related libraries, which we will use to create the interface. Next, we import PyDictionary, which will enable us to get meaning to words that we put in the search box.

## Step 3: Create instances for the packages

In order for us to make use of the two packages we need an instance. In PyDictionary the instance will take in words, while for tkinter it will initialize the tools for usability.

```
dictionary = PyDictionary()  
root=tk()
```

## Step 4: Set Window Dimensions & Title

When you use tkinter, it's important to note that geometry determines the position and size of the screen. In our dictionary to set the size of our window we use the `geometry()`

method.

In our code, we have set the window size to 600x400

and the position of the window to 50px from top and left of the screen. We have also added a title by use of the `title()`

method.

```
# set geometry  
root.title("Dictionary")  
root.geometry("600x400+50+50")
```

## Step 5: Define main function

Our function will help us to make use of other attributes of the PyDictionary

class, such as getting the meaning, synonyms and antonyms of the issued words.



```
def dict():
```

```
    meaning.config(text=dictionary.meaning(word.get())['Noun'][0])
```

```
    synonym.config(text=dictionary.synonym(word.get()))
```

```
    antonym.config(text=dictionary.antonym(word.get()))
```

## Step 6: Add Labels and Buttons

This particular step is where we get to design the interface of our dictionary, that is by setting the fonts, colors, text position etc...

```
# Add labels, buttons and frame
```

```
Label(root, text="My Dictionary", font=(  
    "Poppins, 20 bold"), fg="Orange").pack(pady=20)
```

### # Frame 1

```
frame = Frame(root)  
Label(frame, text="Enter word: ", font=(  
    "Helvetica, 15 bold")).pack(side="left")  
word = Entry(frame, font=("Helvetica, 15 bold"), width=30)  
word.pack()  
frame.pack(pady=10)
```

### # Frame 2

```
frame1 = Frame(root)  
Label(frame1, text="Meaning: ", font=("Aerial, 15 bold")).pack(side=LEFT)  
meaning = Label(frame1, text="", font=("Poppins, 15"))  
meaning.pack()  
frame1.pack(pady=10)
```

### # Frame 3

```
frame2 = Frame(root)  
Label(frame2, text="Synonym: ", font=(  
    "Roboto, 15 bold")).pack(side=LEFT)  
synonym = Label(frame2, text="", font=("Roboto, 15"))  
synonym.pack()  
frame2.pack(pady=10)
```

#### # Frame 4

```
frame3 = Frame(root)
Label(frame3, text="Antonym: ", font=("Helvetica, 15 bold")).pack(side=LEFT)
antonym = Label(frame3, text="", font=("Helvetica, 15"))
antonym.pack(side=LEFT)
frame3.pack(pady=10)
Button(root, text="Search", font=("Helvetica, 15 bold"), command=dict).pack()
```

#### # Frame 4

```
frame3 = Frame(root)
Label(frame3, text="Antonym: ", font=("Helvetica, 15 bold")).pack(side=LEFT)
antonym = Label(frame3, text="", font=("Helvetica, 15"))
antonym.pack(side=LEFT)
frame3.pack(pady=10)
```

### Step 7: Run the App

To execute the app we use the [mainloop\(\)](#)

method. What it does is, it tells Python to run the tkinter

event loop until the user exits it.

# Execute Tkinter

```
root.mainloop()
```

### Step 8: Output

Running the above code will create a display of our dictionary app that will look as shown below with everything set as defined in the code.

## Source code:

```
import json
from difflib import get_close_matches
with open('E:\python files\data.json') as f:
    data = json.load(f)
def translate(word):
    word = word.lower()
    if word in data:
        return data[word]
    elif word.title() in data:
        return data[word.title()]
    elif word.upper() in data:
        return data[word.upper()]
    elif len(get_close_matches(word, data.keys())) > 0:
        print("do you want to find %s" %get_close_matches(word, data.keys())[0])
        decide = input("press y for yes and n for no")
        if decide == "y":
            return data[get_close_matches(word, data.keys())[0]]
        elif decide == "n":
            return("Wrong search!! Please try again")

        else:
            return("Wrong input! Please enter y or n")
    else:
        print("Wrong search!! Please try again")


word = input("Enter the word you want to search")
output = translate(word)
if type(output) == list:
    for item in output:
        print(item)
else:
    print(output)
```

## OUTPUT:

Enter the word you want to search cotton

Fiber obtained from plants of the genus *Gossypium*, used in making fabrics, cordage, and padding and for producing artificial fibers and cellulose.

A shrub of the genus *Gossypium* is known for the soft fibers that protect its seeds.

A screenshot of a web application titled "My Dictionary". The interface is simple and clean, with a light gray background. At the top, the title "My Dictionary" is displayed in a bold, orange font. Below the title, there is a label "Enter word:" followed by a text input field. Underneath the input field, there are three labels: "Meaning:", "Synonym:", and "Antonym:", each followed by a corresponding text area for the results. At the bottom of the form, there is a "Search" button with a black border and text.

## Conclusion:

Now you can create a Dictionary App using Python that can run through the terminal and an interactive GUI. There is no limitation on the design of the app. Therefore, you can redesign it to meet your own specification and style. You can even add other interactivity features to it. It's about time you start using your own dictionary.

Python is a great language that comes with many different features. It offers a structured code, making it easier to understand. With Python being one of the most prominent programming languages in today's day and age, it is important to have a thorough understanding of this programming language.

Python has a large community that tends to develop this programming language. These days, it's common to use Python for FinTech, Data Science, Machine

Learning, and so on. As a result, such giants as Google, Yandex, and Dropbox support this technology and create big project using Python.

Python is easy to learn and powerful programming language as it is known in common parlance, there is nevertheless need of a good introduction and tutorial on the python language

Python supports both function-oriented and structure-oriented programming. It has features of dynamic memory management which can make use of computational resources efficiently. It is also compatible with all popular operating systems and platforms. Hence this language can be universally accepted by all programmers.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.