**MUST**

**MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**Academic Year 2021-25**

**Department: CS/IT**

**Name of Assignment: AI Dictionary**

**Full Name:** Syed Zafran Haider Kazmi

**Roll No.:** FA21-BIT-064

**Section:** “A”

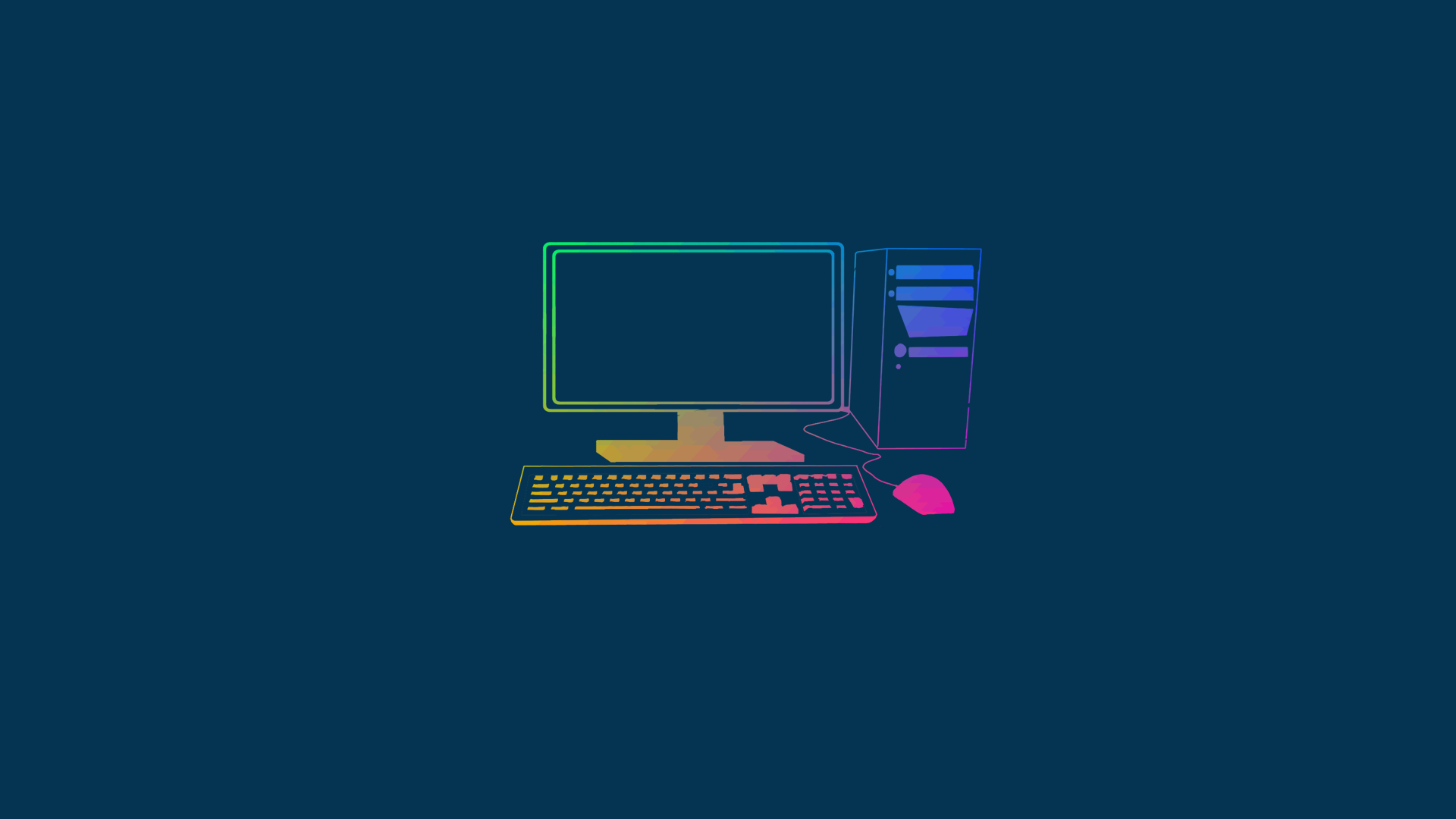
**FIELD;BS IT**

**Semester 5th**

**Subject:** Programming in Python

Teacher Mam Iram;

**Date of Submission:**5/2/2024

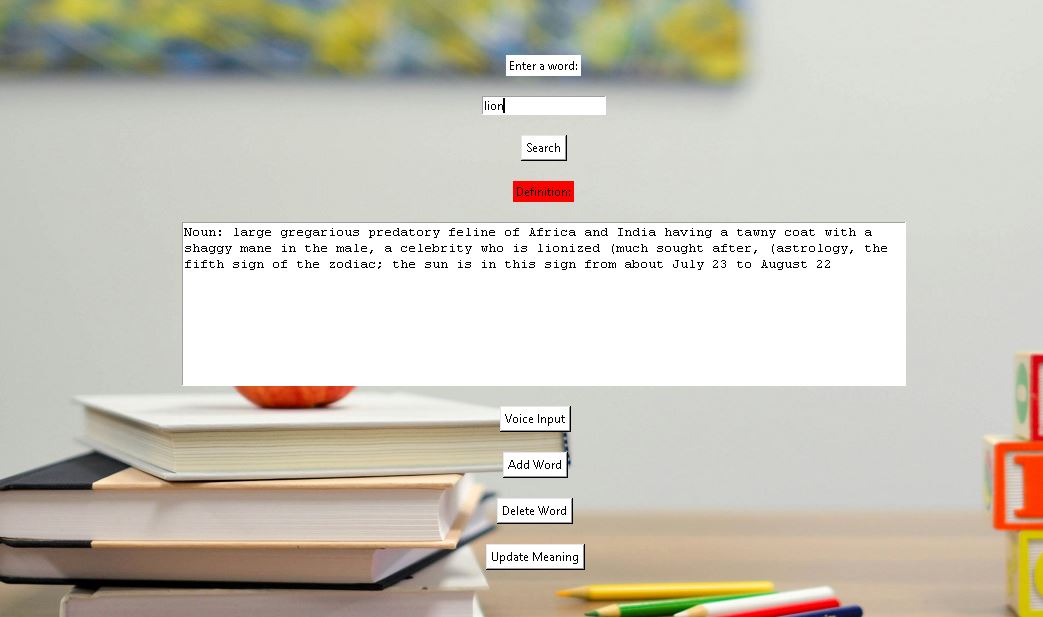


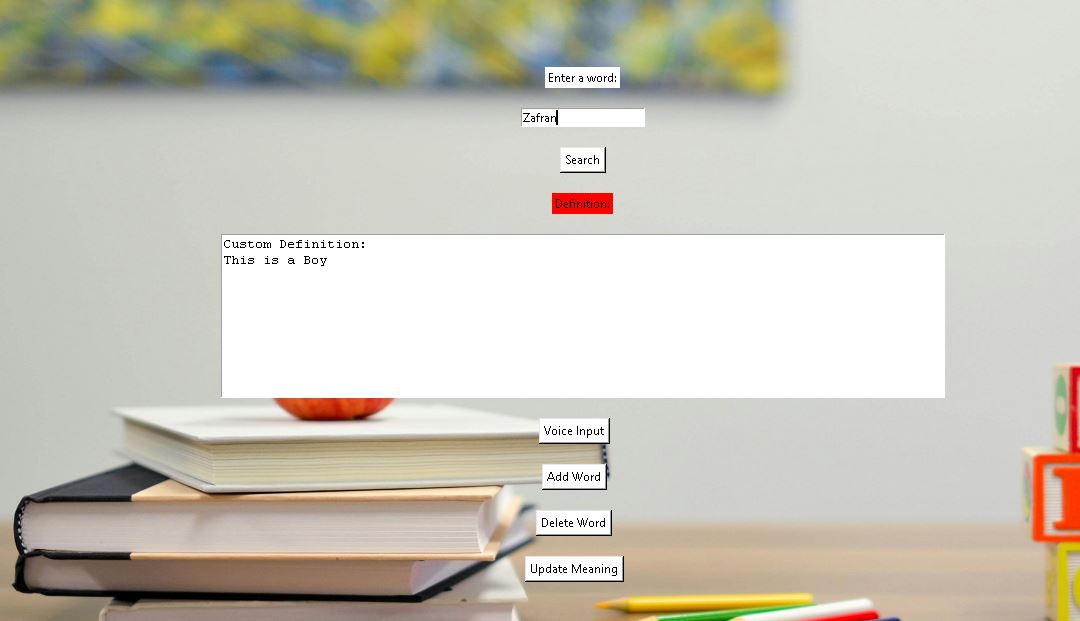
**Q1.** **English AI dictionary. Here are some potential tasks students might undertake:**

1. **Designing the Dictionary Structure: Students could design the data structure to store words and their meanings, perhaps using key-value pairs or other data structures like trees or graphs.**
2. **Implementing Basic Operations: Students could implement basic operations such as adding new words, searching for words, updating meanings, and deleting entries.**
3. **Integrating AI Features: Students might explore integrating AI features such as natural language processing (NLP) for better search results, synonym suggestions, or even voice recognition for input.**

Screnshots

Main Dictionary



****

**Custom Added Name**

**Code**

import tkinter as tk

from tkinter import messagebox

import speech\_recognition as sr

from PyDictionary import PyDictionary

class DictionaryApp:

    def \_\_init\_\_(self, master):

        self.master = master

        self.master.title("English AI Dictionary")

        # Set window size and center the window

        window\_width = 1360

        window\_height = 780

        screen\_width = master.winfo\_screenwidth()

        screen\_height = master.winfo\_screenheight()

        x\_coordinate = (screen\_width - window\_width) // 2

        y\_coordinate = (screen\_height - window\_height) // 2

        self.master.geometry(f"{window\_width}x{window\_height}+{x\_coordinate}+{y\_coordinate}")

        # Load background image

        self.background\_image = tk.PhotoImage(file="back.png")  # Provide the path to your image

        self.background\_label = tk.Label(master, image=self.background\_image)

        self.background\_label.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        self.dictionary = PyDictionary()

        # Initialize custom dictionary

        self.custom\_dictionary = {}

        # Widgets for searching

        self.search\_label = tk.Label(master, text="Enter a word:", bg="white")

        self.search\_label.pack(pady=(100, 10))

        self.search\_entry = tk.Entry(master)

        self.search\_entry.pack(pady=10)

        self.search\_button = tk.Button(master, text="Search", command=self.search\_word, bg="white")

        self.search\_button.pack(pady=10)

        # Widgets for displaying definition

        self.definition\_label = tk.Label(master, text="Definition:", bg="red")

        self.definition\_label.pack(pady=10)

        self.definition\_text = tk.Text(master, height=10, width=90, wrap="word")

        self.definition\_text.pack(pady=10)

        scrollbar = tk.Scrollbar(master, command=self.definition\_text.yview)

        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        self.definition\_text.config(yscrollcommand=scrollbar.set)

        # Widgets for voice input

        self.voice\_button = tk.Button(master, text="Voice Input", command=self.voice\_input, bg="white")

        self.voice\_button.pack(pady=10)

        # Widgets for adding words

        self.add\_button = tk.Button(master, text="Add Word", command=self.create\_add\_word\_window, bg="white")

        self.add\_button.pack(pady=10)

        # Widget for deleting words

        self.delete\_button = tk.Button(master, text="Delete Word", command=self.create\_delete\_word\_window, bg="white")

        self.delete\_button.pack(pady=10)

        # Widget for updating words

        self.update\_button = tk.Button(master, text="Update Meaning", command=self.create\_update\_word\_window, bg="white")

        self.update\_button.pack(pady=10)

    def search\_word(self):

        word = self.search\_entry.get()

        definition = self.dictionary.meaning(word)

        if definition:

            self.definition\_text.delete(1.0, tk.END)

            for pos, mean in definition.items():

                self.definition\_text.insert(tk.END, f"{pos.capitalize()}: {', '.join(mean)}\n")

        elif word in self.custom\_dictionary:

            self.definition\_text.delete(1.0, tk.END)

            self.definition\_text.insert(tk.END, f"Custom Definition:\n{', '.join(self.custom\_dictionary[word])}")

        else:

            messagebox.showerror("Error", f"'{word}' not found in the dictionary.")

    def voice\_input(self):

        r = sr.Recognizer()

        with sr.Microphone() as source:

            print("Speak:")

            try:

                audio = r.listen(source, timeout=5)

                word = r.recognize\_google(audio)

                self.search\_entry.delete(0, tk.END)

                self.search\_entry.insert(0, word)

            except sr.WaitTimeoutError:

                messagebox.showerror("Error", "Timeout waiting for speech")

            except sr.UnknownValueError:

                messagebox.showerror("Error", "Could not understand audio")

            except sr.RequestError as e:

                messagebox.showerror("Error", f"Could not request results from Google Speech Recognition service; {e}")

    def add\_word(self, word, meanings):

        # Add word and meanings to the custom dictionary

        self.custom\_dictionary[word] = meanings

        messagebox.showinfo("Success", f"'{word}' added successfully.")

    def delete\_word(self, word):

        # Delete word from the custom dictionary

        if word in self.custom\_dictionary:

            del self.custom\_dictionary[word]

            messagebox.showinfo("Success", f"'{word}' deleted successfully.")

        else:

            messagebox.showerror("Error", f"'{word}' not found in the custom dictionary.")

    def update\_word(self, word, new\_meanings):

        # Update meanings of a word in the custom dictionary

        if word in self.custom\_dictionary:

            self.custom\_dictionary[word] = new\_meanings

            messagebox.showinfo("Success", f"Meanings of '{word}' updated successfully.")

        else:

            messagebox.showerror("Error", f"'{word}' not found in the custom dictionary.")

    def create\_add\_word\_window(self):

        def add\_word\_to\_dict():

            word = word\_entry.get()

            meanings = meanings\_entry.get("1.0", tk.END).strip().split('\n')

            self.add\_word(word, meanings)

            add\_word\_window.destroy()

        add\_word\_window = tk.Toplevel(self.master)

        add\_word\_window.title("Add Word")

        add\_word\_window.geometry("400x300")

        word\_label = tk.Label(add\_word\_window, text="Enter a word:")

        word\_label.pack(pady=5)

        word\_entry = tk.Entry(add\_word\_window)

        word\_entry.pack(pady=5)

        meanings\_label = tk.Label(add\_word\_window, text="Enter meanings (separated by new lines):")

        meanings\_label.pack(pady=5)

        meanings\_entry = tk.Text(add\_word\_window, height=10, width=30)

        meanings\_entry.pack(pady=5)

        add\_button = tk.Button(add\_word\_window, text="Add", command=add\_word\_to\_dict)

        add\_button.pack(pady=10)

    def create\_delete\_word\_window(self):

        def delete\_word\_from\_dict():

            word = word\_entry.get()

            self.delete\_word(word)

            delete\_word\_window.destroy()

        delete\_word\_window = tk.Toplevel(self.master)

        delete\_word\_window.title("Delete Word")

        delete\_word\_window.geometry("300x150")

        word\_label = tk.Label(delete\_word\_window, text="Enter a word to delete:")

        word\_label.pack(pady=5)

        word\_entry = tk.Entry(delete\_word\_window)

        word\_entry.pack(pady=5)

        delete\_button = tk.Button(delete\_word\_window, text="Delete", command=delete\_word\_from\_dict)

        delete\_button.pack(pady=10)

    def create\_update\_word\_window(self):

        def update\_word\_in\_dict():

            word = word\_entry.get()

            new\_meanings = new\_meanings\_entry.get("1.0", tk.END).strip().split('\n')

            self.update\_word(word, new\_meanings)

            update\_word\_window.destroy()

        update\_word\_window = tk.Toplevel(self.master)

        update\_word\_window.title("Update Meaning")

        update\_word\_window.geometry("400x300")

        word\_label = tk.Label(update\_word\_window, text="Enter a word to update:")

        word\_label.pack(pady=5)

        word\_entry = tk.Entry(update\_word\_window)

        word\_entry.pack(pady=5)

        new\_meanings\_label = tk.Label(update\_word\_window, text="Enter new meanings (separated by new lines):")

        new\_meanings\_label.pack(pady=5)

        new\_meanings\_entry = tk.Text(update\_word\_window, height=10, width=30)

        new\_meanings\_entry.pack(pady=5)

        update\_button = tk.Button(update\_word\_window, text="Update", command=update\_word\_in\_dict)

        update\_button.pack(pady=10)

def main():

    root = tk.Tk()

    app = DictionaryApp(root)

    root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

    main()