Roadmap – „AlphaBet.a v1.0“ – eine effektive Lernhilfe nach der ABC-Methode.

Version 1.0 - Release (Finale Version)

Ziel: Ein Lernprogramm erstellen, dass nach dem ToDo-Manager-Prinzip funktioniert und eine Hilfe zum effektiven Lernen bietet, indem man eine Vokabeldatenbank erzeugt und sich abfragen lassen kann.

1. Funktionen festlegen:

- **Themenmanagement**: Nutzer kann verschiedene Themen erstellen, auswählen und bearbeiten / ändern.
- **Wörterverwaltung**: Hinzufügen, Bearbeiten und Löschen von Wörtern (mit Erklärungen)
- **Speicherung der Vokabeln**: JSON-Datei (vocabulary.json)
- **Quiz-Funktion**: Benutzer können Fragen zu den eingegebenen Wörtern beantworten mit einer Shuffle-Funktion zur zufälligen Abfrage.

2. Gestaltung der Benutzeroberfläche (mit PyQt6)

- **Hauptkomponenten**:
  - **QApplication**: lädt alle anderen GUI-Elemente
  - **QWidget**: GUI-Objekte erstellen, z. B. das Hauptfenster der Anwendung.
  - **QVBoxLayout**: Layout-Manager
  - **QPushButton**: Knöpfe für Useraktionen
  - **QTextEdit**: Mehrzeiliges Eingabefeld für die Erklärungen.
  - **QMessageBox**: Nachrichten / Fehlermeldungen anzeigen
  - **QListWidget**: Liste für eingegebene Wörter, die man auswählen kann.
  - **QComboBox**: Dropdown-Menü zur Auswahl des Themas.
  - **QDialog**: Eröffnen von Dialogfenstern, z. B. zum Bearbeiten der Erklärungen.
  - **QLabel**: Anzeigen von statischem Text oder Informationen in der GUI.
  - **QFormLayout**: Layout festlegen zur Anordnung von Feldern in einem Formular (z.B. beim Bearbeiten von Erklärungstexten).
  - **QInputDialog**: Dialog zur Eingabe von Text.

3. Datenverwaltung

- **Datenstruktur**: Eine Dictionary, um Wörter und Erklärungen pro Thema zu speichern. Beispiel:

```
vokabeln_dict = {
  "Thema1": {
    "Wort1": "Erklärung1",
    "Wort2": "Erklärung2"
  },
  "Thema2": {
    "WortA": "ErklärungA",
    "WortB": "ErklärungB"
  }
}
```

Speichern/Laden von Daten: JSON-Bibliothek, um in eine (JSON)Datei zu schreiben oder daraus zu lesen:

```
import json

def save_data(vokabeln_dict):
  with open('vocabulary.json', 'w', encoding='utf-8') as file:
    json.dump(vokabeln_dict, file, ensure_ascii=False, indent=4)

def load_data():
  try:
    with open('vocabulary.json', 'r', encoding='utf-8') as file:
      return json.load(file)
  except FileNotFoundError:
    return {}
```

4. Quiz-Funktionalität

Wörter shuffeln: Verwendung der random.shuffle() Methode, um die Wörter zufällig anzuordnen:

```
import random


words = list(vocab_dict[topic].items())
random.shuffle(words)
```

Frage stellen: Dialoge, zum Beispiel:

```
reply = QMessageBox.question(self, word, "Möchten Sie die Erklärung sehen?", QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)
```

5. Fehlerbehebung und Optimierung

- **Ziel:** Verbesserung der Stabilität und Benutzerfreundlichkeit
    - **Fehlerüberprüfung**: Implementierung von Fehlerbehandlungsmechanismen, um Abstürze zu verhindern, z.B. durch Verwendung von try-except-Blöcken:

```
try:
  # Der Code, der Fehler verursachen könnte
except Exception as e:
  QMessageBox.critical(self, "Fehler", str(e))
```

6. Dokumentation

- **Roadmap:** Detaillierte Entwicklungsbeschreibung mit Erläuterungen.
- **README.md**: Informationen zur Installation, Nutzung und Funktionalität der Anwendung.
- **CHANGELOG.md** (für zukünftige Versionen): Liste von Änderungen, die implementiert wurden oder geplante Änderungen am Programm.

7. Veröffentlichung (Version 1.0)

- **Ziel:** Projekt starten und die Bibliothek füllen mit Vokabeln.
    - **Github-Repository**: Hochladen aller Dateien auf GitHub, inkl. README.md und .gitignore.
    - **Aufbewahrungsrichtlinien**: Sicherstellen, dass sensible oder nicht benötigte Dateien (wie venv) in der .gitignore enthalten sind.