Aplikacja w Pythonie. Celem zajęć jest wprowadzenie do modułu Tkinter do tworzenia interfejsu użytkownika oraz modułu SQLite3 do obsługi bazy danych. Scenariusz zakłada, że uczniowie umieją podstawy takie jak tworzenie zmiennych czy metod, ale nie umieją SQL, więc na wstępie omawiana jest podstawowa składnia tego języka.

Fiszki

Aplikacja do nauki słówek

Ćwiczenie 0 - wprowadzenie do SQL:

Chcemy przedstawić uczniom tylko najważniejsze informacje, ogólnie o co chodzi, czym są bazy i do czego możemy ich używać.

SQL to język programowania używany do zarządzania danymi, na przykład danymi o klientach czy produktach w sklepie internetowym. Te dane potrzebują miejsca, w którym będą przechowywane, nazywamy to "baza danych". Baza danych składa się z tabel, a każda tabela zawiera wiersze i kolumny. Każdy wiersz reprezentuje jeden rekord (np. dane jednego klienta), a kolumny to właściwości tego rekordu (np. imię, nazwisko, adres).

Można narysować to na tablicy właśnie w taki sposób.

Sqlite3 jest lekką, bez serwerową bazą danych, alternatywą dla przechowywania danych w pliku.

W naszej aplikacji do nauki słówek będziemy używali bardzo prostego modelu:

mamy bazę „fiszki.db”, w tej bazie mamy jedną tabelę „words” przechowującą słówka do nauki wraz z tłumaczeniem. Tak więc kolumnami w danym wierszu będą: słowo, tłumaczenie.

Words
word translation

"CREATE TABLE IF NOT EXISTS words (word TEXT PRIMARY KEY, translation TEXT)"

If not exists sprawdza czy istnieje taka tabela. Text to typ, primary key oznacza, że dane słowo nie będzie mogło się powtarzać w naszej bazie, tak jak na przykład nie powtarzają się numery PESEL i jednoznacznie identyfikują osobę, tak u nas słowo „apple” będzie wskazywało tylko na „jabłko”. *Można wytłumaczyć tutaj, iż jest to tylko i wyłącznie nasze założenie na potrzeby tej aplikacji, można wprowadzić indeks i dopuścić do kilku tłumaczeń słowa (tak jak to funkcjonuje często w rzeczywistości).*

A jak wygląda dodawanie danych?

"INSERT INTO words (word, translation) VALUES (?, ?)"

W miejsce “?” podajemy już konkretne dane np. (apple, jabłko)

Możemy też dane usuwać, musimy określić z jakiej tabeli „FROM words” i co chcemy usunąć „WHERE word=apple”:

"DELETE FROM words WHERE word=?"

Edytowanie danych ma lekko inną konstrukcję, „SET” ustawia nowe tłumaczenie, a „WHERE” wskazuje o jakie słowo nam chodzi:

"UPDATE words SET translation=? WHERE word=?"

Zostało już tylko odczytywanie danych, gwiazdka „*” oznacza że chcemy wyświetlić wszystkie kolumny, można też wypisać je po kolei, u nas są tylko dwie (word, translation):

"SELECT * FROM words"

Ćwiczenie 1

Importujemy biblioteki:

```
import tkinter as tk
from tkinter import messagebox
import sqlite3
```

Teraz definiujemy potrzebne metody do tworzenia tabeli i dodawania danych

```
def create_table():
    with sqlite3.connect("fiszki.db") as conn:
        cursor = conn.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS words (word TEXT PRIMARY KEY, translation TEXT)")


def add_word(word, translation):
    with sqlite3.connect("fiszki.db") as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO words (word, translation) VALUES (?, ?)", (word, translation))
        conn.commit()
```

Tworzymy połączenie z bazą „fiszki.db”, znajdującą się lokalnie w naszej aplikacji oraz ustawiamy kursor potrzebny do wykonywania operacji na bazie. Polecenia wykonujemy za pomocą języka SQL. Na koniec zapisujemy zmiany.

Można wspomnieć uczniom o tym, że warto od początku używać angielskiego nazewnictwa,

aczkolwiek jeśli ktoś chce nazwać po polsku to nie jest to błąd.

Ćwiczenie 2

Do stworzenia aplikacji okienkowej wystarcza nam te linijki kodu i dzieci mogą przetestować:

```
from tkinter import Tk
root = Tk()
root.mainloop()
```

Lecz my chcemy dodać obsługę przycisków, więc najpierw będziemy potrzebowali kilku metod.

Chcemy sczytać słowo podane przez użytkownika i sprawdzić czy zgadza się z tłumaczeniem zapisanym w bazie. Używamy funkcji lower() na stringu aby uniknąć sytuacji, a której program nie akceptuje tłumaczenia ze względu na wielkie litery. Nie obsługujemy za to sytuacji gdy użytkownik nie używa polskich znaków tam, gdzie są potrzebne. Tak więc „jabłko” będzie poprawne, lecz „jablko” już nie i trzeba na to uważać. Ustawiamy odpowiedź, którą otrzyma użytkownik w przypadku poprawnej i niepoprawnej odpowiedzi, a także kolor czcionki.

```
def check_translation():
    translation = entry_translation.get()
    if translation.lower() == current_word[1].lower():
        label_result.config(text="Brawo!", fg="green")
    else:
        label_result.config(text=f"Niepoprawna odpowiedź. Prawidłowe tłumaczenie to: {current_word[1]}", fg="red")
```

Chcemy być w stanie usuwać słowo z bazy, gdy się już go nauczymy tak aby nie powtarzać wciąż tego samego. Po naciśnięciu przycisku odpowiedzialnego za usuwanie dostaniemy komunikat w wyskakującym oknie, aby upewnić się, że chcemy modyfikować bazę. Po usunięciu pojawia nam się kolejna fiszka automatycznie.

```
def next_word():
    load_next_word()
    label_result.config(text="")


def remove_word():
    if messagebox.askyesno("Usuwanie", "Czy na pewno chcesz to zrobić?"):
        with sqlite3.connect("fiszki.db") as conn:
            cursor = conn.cursor()
            cursor.execute("DELETE FROM words WHERE word=?", (current_word[0],))
            conn.commit()
        next_word()
```

Gdy zauważymy błąd w tłumaczeniu fajnie by było móc to od razu poprawić, dodamy więc taką funkcjonalność:

```
def update_word():
    new_translation = entry_translation.get()
    with sqlite3.connect("fiszki.db") as conn:
        cursor = conn.cursor()
        cursor.execute("UPDATE words SET translation=? WHERE word=?", (new_translation, current_word[0]))
        conn.commit()
    label_result.config(text="Fiszka została poprawiona.")
    load_next_word()
```

Podobnie jak poprzednio pobieramy słowo podane przez użytkownika w polu tekstowym i ustawiamy nowe tłumaczenie. Wyświetlamy komunikat o udanej edycji. Pobieramy kolejne słowo.

Dodamy też możliwość przeglądania słów w naszej bazie. Pomoże nam w tym cursor.fetchall() zwracający wszystkie wiersze w postaci listy krotek.

Krotka w Pythonie jest sekwencją wartości podobnie jak lista, lecz są niezmienne i mogą zawierać wartości różnych typów.

Słowa pojawią się w nowym oknie, tworzymy browse_window i ustawiamy tytuł.

```
def browse_database():
    with sqlite3.connect("fiszki.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM words")
        data = cursor.fetchall()
    browse_window = tk.Toplevel(root)
    browse_window.title("Przeglądanie bazy danych")
    for idx, word_data in enumerate(data, start=1):
        tk.Label(browse_window, text=f"{idx}. {word_data[0]} - {word_data[1]}").pack()
```

Ostatnią metodą będzie pobieranie kolejnego słowa, „ORDER BY” służy do określenia porządku wyświetlenia, może to być na przykład porządek rosnący lub malejący (ASC, DESC), a w naszym przypadku będzie to losowe pobieranie jednego słowa.

```
def load_next_word():
    global current_word
    with sqlite3.connect("fiszki.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM words ORDER BY RANDOM() LIMIT 1")
        current_word = cursor.fetchone()
    label_word.config(text=current_word[0])
    entry_translation.delete(0, tk.END)
```

Ćwiczenie 3

Implementujemy elementy interfejsu użytkownika. Tworzymy obiekt klasy Tk, który jest najważniejszy w kontekście aplikacji okienkowej, do niego będziemy dodawać elementy interfejsu.

Aby dodać tekst używamy klasy Label. Jako pierwszy argument podajemy gdzie dany element będzie umieszczony czyli nasz root. Można przetestować różne czcionki. Na koniec używamy pack() aby określić położenie elementu.

```
root = tk.Tk()
root.title("Fiszki")
create_table()
current_word = None


label_word = tk.Label(root, text="", font=("Courier New", 20))
label_word.pack(pady=20)

entry_translation = tk.Entry(root, font=("Helvetica", 16))
entry_translation.pack(pady=10)

label_result = tk.Label(root, text="", font=("Arial", 14))
label_result.pack(pady=5)
```

Dodajemy przyciski na podobnej zasadzie: command jest to działanie jakie chcemy wykonać po naciśnięciu.

```
button_check = tk.Button(root, text="Sprawdź", command=check_translation)
button_check.pack(pady=5)

button_next = tk.Button(root, text="Następne słówko", command=next_word)
button_next.pack(pady=5)

button_remove = tk.Button(root, text="Już umiem, usuń z bazy", command=remove_word)
button_remove.pack(pady=5)

button_update = tk.Button(root, text="Popraw fiszkę", command=update_word)
button_update.pack(pady=5)

button_browse = tk.Button(root, text="Przeglądaj bazę", command=browse_database)
button_browse.pack(pady=5)
```

Ostatnią już rzeczą będzie dodanie przykładowych danych:

```
add_word("apple", "jabłko")
add_word("dog", "pies")
add_word("cat", "kot")
add_word("red", "czerwony")
add_word("green", "zielony")
add_word("blue", "niebieski")

load_next_word()
root.mainloop()
```

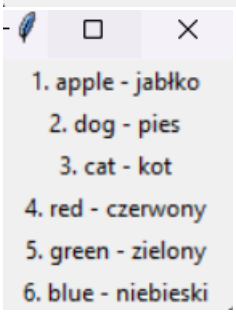
Musimy pamiętać, że gdy już raz stworzymy bazę i dodamy te dane to przy następnym uruchomieniu one już tam będą więc będzie trzeba usunąć tą część kodu z metodą add_word(), ponieważ baza nie pozwoli nam na dodanie kolejnego „apple” itd.

```
    add_word("apple", "jabłko")
  File "C:\Users\wikto\Desktop\appmatplaneta\main.py", line 15, in add_word
    cursor.execute("INSERT INTO words (word, translation) VALUES (?, ?)", (word, translation))
sqlite3.IntegrityError: UNIQUE constraint failed: words.word
```

Ćwiczenie 4

Testujemy naszą aplikację, dzieci mogą dodawać słowa i bawić się fiszkami (lub używać ich do nauki)


Fiszki
apple
jablko
Niepoprawna odpowiedź. Prawidłowe tłumaczenie to: jabłko
Sprawdź
Następne słówko
Już umiem, usuń z bazy
Popraw fiszkę
Przeglądaj bazę


1. apple - jabłko
2. dog - pies
3. cat - kot
4. red - czerwony
5. green - zielony
6. blue - niebieski