**Código para Qwen con prompts resumidos**

"""

Quiz matemático + feedback con Qwen-3-30B-A3B en LM Studio

Requisitos:

    pip install requests tkinter

Estructura del JSON:

{

  "preguntas": [

      {"pregunta": "Factoriza x²-9", "respuesta": "(x-3)(x+3)"},

      ...

  ]

}

Guarda el fichero como preguntas.json y ejecuta: python quiz\_math.py

"""

import json

import random

import re

import requests

import tkinter as tk

from pathlib import Path

from tkinter import scrolledtext

# ---------------- parámetros fáciles de ajustar ----------------------

LM\_URL      = "http://localhost:1234/v1/chat/completions"

MODEL\_NAME  = "qwen3-30b-a3b"

JSON\_FILE   = Path("preguntas.json")

N\_QUESTIONS = 2     # número de preguntas que se mostrarán por intento

VIDEO\_LINK  = "https://www.youtube.com/watch?v=dmUjA2V\_vOQ"  # video breve (~3 min)

# ---------------------------------------------------------------------

# ---------------- utilidades -----------------------------------------

def cargar\_preguntas() -> list[dict]:

    """Carga las preguntas desde el JSON."""

    with JSON\_FILE.open(encoding="utf-8") as f:

        return json.load(f)["preguntas"]

def normaliza(expr: str) -> list[str]:

    """

    Canoniza la respuesta para compararla.

    Convierte todo a minúsculas, quita espacios y paréntesis exteriores

    y ordena los factores.

    """

    expr = expr.lower().replace(" ", "").strip("()")

    return sorted(expr.split(")("))  # p. ej. (x-3)(x+3) → ["(x+3", "x-3)"]

def limpia(raw: str) -> str:

    """Elimina marcas internas del modelo (p. ej. <think>)."""

    raw = re.sub(r"<think>.\*?</think>", "", raw, flags=re.DOTALL)

    raw = re.sub(r"[\\$\*#\[\]]", "", raw)

    return raw.strip()

def llama\_llm(prompt: str) -> str:

    """Envía el prompt al modelo Qwen alojado en LM Studio y devuelve la respuesta limpia."""

    payload = {

        "model": MODEL\_NAME,

        "messages": [

            {"role": "system",

             "content": "Eres un tutor experto en matemáticas."},

            {"role": "user", "content": prompt}

        ],

        "temperature": 0.5,

        "max\_tokens": 768

    }

    try:

        r = requests.post(LM\_URL, json=payload, timeout=60)

        r.raise\_for\_status()

        return limpia(r.json()["choices"][0]["message"]["content"])

    except Exception as e:

        return f"⚠️ Error al conectar con LM Studio: {e}"

# ---------------------------------------------------------------------

class QuizApp:

    def \_\_init\_\_(self):

        # selecciona N\_QUESTIONS aleatorias

        self.preguntas = random.sample(cargar\_preguntas(), N\_QUESTIONS)

        self.errores: list[tuple[int, str]] = []  # (índice, respuesta\_usuario)

        self.puntos = 0

        # ---------- GUI ----------

        self.root = tk.Tk()

        self.root.title("Quiz de Álgebra – Diferencia de cuadrados")

        self.root.geometry("840x540")

        tk.Label(self.root,

                 text="Responde y pulsa «Evaluar»",

                 font=("Arial", 14, "bold")).pack(pady=8)

        self.frames, self.entries, self.tags = [], [], []

        for idx, q in enumerate(self.preguntas, 1):

            fr = tk.Frame(self.root)

            fr.pack(anchor="w", pady=4)

            tk.Label(fr,

                     text=f"{idx}. {q['pregunta']}",

                     font=("Arial", 12)).pack(side="left")

            ent = tk.Entry(fr, width=25, font=("Arial", 12))

            ent.pack(side="left", padx=6)

            btn = tk.Button(fr,

                            text="Evaluar",

                            command=lambda i=idx-1: self.evaluar(i))

            btn.pack(side="left")

            tag = tk.Label(fr, text="", font=("Arial", 12, "bold"))

            tag.pack(side="left", padx=6)

            self.frames.append(fr)

            self.entries.append(ent)

            self.tags.append(tag)

        # bloque de resultado y recomendaciones

        self.lbl\_score = tk.Label(self.root, font=("Arial", 13, "bold"))

        self.btn\_rec   = tk.Button(self.root,

                                   text="Mostrar recomendaciones",

                                   command=self.recomendar,

                                   bg="green", fg="white")

        self.txt\_rec   = scrolledtext.ScrolledText(self.root,

                                                   width=100, height=10,

                                                   font=("Arial", 11),

                                                   state="disabled")

    # ---------------- lógica ------------------------------------------

    def evaluar(self, i: int):

        """Evalúa la respuesta de la pregunta i."""

        usr  = self.entries[i].get()

        good = self.preguntas[i]["respuesta"]

        ok   = normaliza(usr) == normaliza(good)

        self.tags[i].configure(text=("✅ Correcto" if ok else "❌ Incorrecto"),

                               fg=("green" if ok else "red"))

        if ok:

            self.puntos += 1

        else:

            self.errores.append((i, usr))

        # Desactiva controles de la fila

        self.entries[i].config(state="disabled")

        self.frames[i].children["!button"].config(state="disabled")

        # Si todas las preguntas se evaluaron, muestra puntaje

        if all(self.frames[j].children["!button"]["state"] == "disabled"

               for j in range(N\_QUESTIONS)):

            self.mostrar\_puntaje()

    def mostrar\_puntaje(self):

        pct = 100 \* self.puntos / N\_QUESTIONS

        self.lbl\_score.config(text=f"Puntaje final: {pct:.0f}%")

        self.lbl\_score.pack(pady=8)

        self.btn\_rec.pack(pady=4)

    def recomendar(self):

        if not self.txt\_rec.winfo\_ismapped():

            prompt = self.\_genera\_prompt()

            feedback = llama\_llm(prompt)

            self.txt\_rec.config(state="normal")

            self.txt\_rec.insert("1.0", feedback)

            self.txt\_rec.config(state="disabled")

            self.txt\_rec.pack(padx=10, pady=6, fill="both", expand=True)

    def \_genera\_prompt(self) -> str:

        """

        Construye el prompt para el modelo dependiendo del desempeño.

        - Si todo está bien: felicita y sugiere más retos.

        - Si todo está mal: repasa la teoría de diferencia de cuadrados y

          enlaza a un video.

        - En casos intermedios: retroalimentación detallada por error.

        """

        # 1. Todas correctas

        if not self.errores:

            return (

                "Todas las respuestas fueron correctas. "

                "Felicita al estudiante y sugiere ejercicios más retadores. "

                "Concluye con /no\_think."

            )

        # 2. Todas incorrectas

        if len(self.errores) == N\_QUESTIONS:

            return (

                "Todas las respuestas fueron incorrectas.\n\n"

                "Recuerda al estudiante la teoría de la diferencia de cuadrados:\n"

                "La forma general es a^2 - b^2 = (a - b)(a + b). Explica claramente\n"

                "cómo identificar los cuadrados perfectos, extraer las raíces y\n"

                "aplicar la fórmula paso a paso.\n\n"

                "Incluye un ejemplo completo resuelto (por ejemplo, x² - 25) con\n"

                "explicación por pasos numerados.\n\n"

                f"Al final, sugiere ver este video breve: {VIDEO\_LINK}\n"

                "NO uses Markdown ni LaTeX. Concluye con /no\_think."

            )

        # 3. Caso mixto: algunos aciertos, algunos errores

        detalle = "\n".join(

            (f"Pregunta {idx+1}: {self.preguntas[idx]['pregunta']}\n"

             f"Respuesta escrita: {resp}\n"

             f"Respuesta correcta: {self.preguntas[idx]['respuesta']}")

            for idx, resp in self.errores

        )

        return (

        "Se detectaron errores al factorizar diferencias de cuadrados.\n\n"

        "Para \*\*cada error\*\* escribe un bloque con este formato EXACTO (sin Markdown):\n\n"

        # ──────────────── nuevo formato ────────────────

    "Se tienen errores en la Pregunta X\n"

"Respuesta dada: ...\n"

"Descripción de los errores.\n"

"... (explicación aquí) ...\n\n"

"Cómo debió resolverse.\n"

"... (resolución aquí) ...\n\n"

"Consejos prácticos.\n"

"... (consejos aquí) ...\n"

# añade un separador visual

        "----  (deja una línea en blanco después de cada bloque)  ----\n\n"

        # ───────────────────────────────────────────────────────────

        f"{detalle}\n\n"

        "- Concluye con /no\_think."

    )

    # ------------------------------------------------------------------

    def run(self):

        self.root.mainloop()

# ---------------- lanzamiento ----------------------------------------

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

    QuizApp().run()