Prática de POO com Bibliotecas, APIs e Framework (python)

Em dupla!!

O Projeto: World Guessing Game (Ranking Edition)

A lógica: O programa busca um país aleatório via API. O aluno deve adivinhar o nome do país com base em uma dica (como a capital ou o continente). No final, o sistema calcula a pontuação, aplica um bônus matemático e salva o recorde no banco de dados.

Como os 4 Pilares se encaixam:

1. **API (`Rest Countries`):**
   - **URL:** `https://restcountries.com/v3.1/all`
   - **Uso:** O código baixa uma lista de países e escolhe um aleatoriamente. Ele extrai o nome (resposta), a capital (dica) e a população.
2. **Biblioteca (`math`):**
   - **Uso:** Calcular a pontuação final. Exemplo: `pontos = math.floor(1000 / tempo_gasto)`. Se o país tiver uma população pequena, o aluno ganha um multiplicador de dificuldade usando `math.log`.
3. **Biblioteca (`datetime`):**
   - **Uso:** Cronometrar quanto tempo o aluno levou para responder (diferença entre o momento em que a dica apareceu e o momento do palpite).
4. **Banco de Dados (`sqlite3`):**
   - **Uso:** Uma tabela `ranking` com colunas: `jogador`, `pontuacao`, `data_hora`.
5. **Framework (`tkinter`):**
   - **Uso:** Uma tela com a dica, um campo de entrada para o palpite e uma tabela (`Treeview`) para mostrar os 5 melhores jogadores do banco.

Abaixo está o roteiro lógico que os alunos devem seguir para construir o **World Guessing Game**.

1. Arquitetura das Classes

O sistema deve ser dividido em responsabilidades claras:

- **Classe `BancoDados`**: Responsável por criar a conexão, a tabela e os métodos de `INSERT` (salvar recorde) e `SELECT` (buscar o Top 5).
- **Classe `ServicoAPI`**: Responsável por fazer o `requests.get`, tratar o JSON e retornar um país aleatório com seus dados (Nome, Capital, População).
- **Classe `JogoApp` (Interface)**: A classe principal que herda de `tk.Tk` ou gerencia o `root`. Ela fará a ponte entre a API, o Banco e os cálculos do `math` e `datetime`.

---

2. Passo a Passo do Funcionamento Logístico

Fase 1: Persistência (Banco de Dados)

1. **Inicialização**: Ao abrir o programa, o código deve verificar se o arquivo `.db` existe. Se não, cria a tabela de ranking.
2. **Método Salvar**: Deve receber o nome do aluno e a pontuação, capturar a data atual com `datetime` e gravar no SQLite.
3. **Método Listar**: Deve retornar apenas os 5 maiores registros ordenados pela pontuação.

Fase 2: Consumo de Dados (API)

1. **Requisição**: O código acessa o endpoint da API de países.
2. **Tratamento**: Como a API retorna centenas de países, o código deve usar a biblioteca `random` para selecionar apenas um.
3. **Filtragem**: Deve-se extrair o nome comum do país (para ser a resposta correta) e a capital (para ser a dica exibida na tela).

Fase 3: A Rodada do Jogo (Lógica de Negócio)

1. **Início da Rodada**: Assim que o país é sorteado e a dica aparece, o código captura o "tempo de agora" usando `datetime.now()`.
2. **Processamento do Palpite**: Quando o usuário clica no botão "Chutar":
   - Captura o novo `datetime.now()` e calcula a diferença (em segundos) entre o início e o fim.
   - **Uso do Math**: Se o palpite estiver correto, aplica uma fórmula matemática. Exemplo: `1000 dividido pelo tempo em segundos`. Quanto mais rápido, mais pontos.
   - **Bônus**: Pode-se usar `math.log` ou `math.sqrt` sobre a população do país (dados da API) para dar mais pontos a países menores/mais difíceis.
3. **Finalização**: Se acertou, o código chama o método da classe de Banco de Dados para salvar o recorde e limpa o campo para uma nova rodada.

Fase 4: Interface Gráfica (Framework Tkinter)

1. **Construção**: Criar os Widgets (Labels para dicas, Entry para o texto, Button para a ação).
2. **O Componente Treeview**: Este é o ponto crucial. O aluno deve configurar uma tabela (grid) que se auto-atualiza toda vez que um novo recorde é inserido no banco.
3. **Loop Principal**: O `mainloop()` mantém a janela aberta e o programa aguardando o evento de clique do aluno.

⚠️⚠️⚠️ Antes da Entrega:

Aqui está o checklist de testes que cada aluno deve realizar antes de entregar o projeto.

📋 Guia de Testes e Critérios de Aceite

1. Teste de Inicialização e Persistência (Banco de Dados)

Este teste garante que a Classe de Banco de Dados está criando a estrutura correta.

- **Ação:** Exclua o arquivo `.db` da pasta e execute o código.
- **Resultado Esperado:** O programa deve abrir sem erros e um novo arquivo `.db` deve aparecer na pasta do projeto.
- **Se falhar:** A lógica de `CREATE TABLE IF NOT EXISTS` dentro do método `__init__` da classe de banco está errada.

2. Teste de Consumo de Dados (API)

Este teste valida se a Classe de Serviço de API está tratando o JSON corretamente.

- **Ação:** Observe a dica (Capital) que aparece na tela.
- **Resultado Esperado:** A dica deve ser uma String legível (Ex: "Paris") e não uma estrutura de lista ou dicionário (Ex: `['Paris']` ou `{'capital': 'Paris'}`).
- **Se falhar:** O aluno esqueceu de acessar o índice `[0]` da lista de capitais que a API de países retorna.

3. Teste de Cálculo de Pontuação (Math + Datetime)

Este teste valida a precisão da lógica de negócio.

- **Ação:** Responda corretamente o país em menos de 2 segundos. Depois, jogue novamente e responda corretamente após 10 segundos.
- **Resultado Esperado:** A pontuação do primeiro acerto deve ser obrigatoriamente maior que a do segundo acerto.

- **Se falhar:** O cálculo que utiliza a diferença entre os objetos `datetime` não está sendo aplicado à fórmula da biblioteca `math`.

4. Teste de Fluxo de Interface (Framework Tkinter)

Este teste valida a Inversão de Controle e a experiência do usuário.

- **Ação:** Digite o nome correto e clique no botão de envio.
- **Resultado Esperado:** 1. Uma mensagem de sucesso deve aparecer.
  2. O campo de entrada (Entry) deve ser limpo automaticamente.
  3. Uma nova dica de capital deve aparecer imediatamente sem precisar reiniciar o programa.
- **Se falhar:** O método de resposta não está chamando o método de "nova rodada" ao final da execução.

5. Teste de Ranking em Tempo Real (Integração Total)

Este teste valida se todas as classes estão conversando entre si.

- **Ação:** Termine uma partida vitoriosa e observe a tabela (Treeview).
- **Resultado Esperado:** O nome e a pontuação que você acabou de obter devem aparecer no topo (ou entre os 5 primeiros) da tabela visual na interface, sem que você precise fechar e abrir o jogo.
- **Se falhar:** O método que atualiza a Treeview não está executando um novo `SELECT` no banco após o `INSERT`.

> **Como enviar:**
>
> 1. Crie um novo repositório com o nome “w**orld_guessing_game” (público);**
> 2. Suba todo o código respeitando os principios orientado a objeto (arquivos para cada classe e um arquivo para main);
> 3. Envie o link do repositorio na atividade e clique em concluir;
> 4. Ambos da dupla devem enviar o código para seu repositório, ou seja, o código pode ser produzido junto, mas o envio para o repositório deve ser individual;
>
> **Critério de Portfólio:** Construa um código limpo, profissional e autoral. Siga os padrões de boas práticas de Python e evite o uso de comentários superficiais gerados por IA. O foco aqui é demonstrar sua maturidade técnica para empresas, criando uma base sólida para o seu portfólio.