Criando Seu Primeiro Agente de IA

Slide 1: Introdução - O Que Vamos Construir?

- **Objetivo:** Este guia detalha a configuração do ambiente e o desenvolvimento de um agente de IA simples.
- **Ferramentas Essenciais:**
  - **WSL (Windows Subsystem for Linux):** Para um ambiente Linux no Windows.
  - **UV:** Um gerenciador de pacotes e ambientes Python moderno e rápido.
  - **Gradio:** Para criar uma interface web fácil para o agente.
  - **Gemini API:** A inteligência artificial por trás do nosso agente.

Slide 2: Preparando o Ambiente Windows (WSL & Hyper-V)

- **1. Instale o PowerShell:** Certifique-se de que você tem o PowerShell no seu sistema Windows.
- **2. Instale o WSL (Windows Subsystem for Linux):** Siga as instruções oficiais da Microsoft para instalar o WSL no seu computador.
- **3. Habilite o Hyper-V (Essencial para WSL2):**
  - Abra o **PowerShell como Administrador**.
  - Execute o seguinte script para habilitar os recursos do Hyper-V:

```
pushd "%~dp0"
dir /b %SystemRoot%\servicing\Packages\*Hyper-V*.mum >hyper-v.txt
for /f %%i in ('findstr /i . hyper-v.txt 2^>nul') do dism /online /norestart /add-package:"%SystemRoot%\servicing\Packages\%%i"
del hyper-v.txt
Dism /online /enable-feature /featurename:Microsoft-Hyper-V -All /LimitAccess /ALL
```

```
pause
```

- **Reinicie sua máquina** após a conclusão da instalação do Hyper-V.

Slide 3: Instalação e Configuração do UV (no WSL)

- **1. Instale o UV:**
  - Abra o terminal **WSL (Linux)**.
  - Execute o comando para instalar o UV:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

- **2. Crie um Novo Usuário (Recomendado):**
  - **Por segurança, evite trabalhar como** `root`**.** Crie um usuário padrão.
  - No terminal WSL, crie um novo usuário (substitua `seu_nome_de_usuario`):

```
adduser seu_nome_de_usuario
```

  - Siga as instruções para definir a senha e informações opcionais (pressione **Enter** para pular campos como "Room Number").
  - Mude para o diretório home do seu novo usuário:

```
cd /home/seu_nome_de_usuario
```

Slide 4: Preparando o Projeto Python

- **1. Crie o Diretório do Projeto:**
  - No terminal WSL (no diretório do seu usuário), crie e entre na pasta do projeto:

```
mkdir meu_agente_ia
cd meu_agente_ia
```

- *Sugestão: Use um nome significativo para a pasta, como `meu_agente_ia` ou `projeto_gemini`.*

- **2. Crie um Ambiente Virtual com Python 3.12:**
  - Dentro da pasta do projeto (`meu_agente_ia`):

```
uv venv --python 3.12
```

- **3. Ative o Ambiente Virtual:**
  - Sempre que for trabalhar no projeto, ative o ambiente virtual:

```
source .venv/bin/activate
```

- **4. Adicione as Dependências Essenciais:**
  - Com o ambiente virtual ativado, adicione os pacotes:

```
uv add gradio openai-agents python-dotenv
```

  - *O `uv` irá instalar os pacotes e criar (ou atualizar) o arquivo `pyproject.toml` no seu projeto.*

- **5. Opcional: Gerenciar `requirements.txt`**
  - Para exportar dependências: `uv pip freeze > requirements.txt`
  - Para instalar em outro ambiente: `uv pip install -r requirements.txt`

Slide 5: Configuração da API Gemini

- **1. Obtenha Sua Chave de API:**
  - Acesse o Google AI Studio: **https://aistudio.google.com/**
  - Navegue até a seção de **chaves de API** e crie uma nova.
  - **Copie sua chave API** gerada.

- **2. Crie o Arquivo `.env`:**
  - Na **pasta raiz do seu projeto (`meu_agente_ia`)**, crie um arquivo chamado `.env`.
  - Cole sua chave API dentro dele no formato:

    ```
    TOKEN="SUA_CHAVE_DE_API_GEMINI_AQUI"
    ```

  - **SEGURANÇA:** Nunca compartilhe este arquivo ou sua chave API publicamente!

Slide 6: O Código do Seu Agente de IA (`main.py`)

- **Crie `main.py`:** Na pasta do seu projeto (`meu_agente_ia`), crie um arquivo chamado `main.py` e adicione o seguinte código:

```
import os
import gradio as gr
from agents import Agent, OpenAIChatCompletionsModel, Runner, set_traci
ng_disabled, AsyncOpenAI
from dotenv import load_dotenv

# Carrega as variáveis de ambiente do arquivo .env
load_dotenv()
# Desabilita o rastreamento (tracing) para o agente.
set_tracing_disabled(disabled=True)

# URL do proxy para acessar o Gemini através da interface compatível com
OpenAI
proxy_url = "https://generativelanguage.googleapis.com/v1beta/openai/"
# Inicializa o cliente AsyncOpenAI, apontando para o proxy e usando a cha
ve da API.
client = AsyncOpenAI(base_url=proxy_url, api_key=os.getenv("TOKEN"))

# Define o agente de IA "Instrutor"
agent = Agent(
    # Usa o modelo Gemini 1.5 Flash, um modelo atualizado e eficiente.
    model=(OpenAIChatCompletionsModel(model="gemini-1.5-flash", opena
```

```
i_client=client)),
    name="Instrutor",
    instructions="""
Você é um agente com propósito de ajudar usuários a criar agentes de IA
Seja educado e prestativo

Caso o usuário se desvie da finalidade lembre a ele que você é uma ferram
enta de estudo
""",
    tools=[] # O agente não tem ferramentas configuradas ainda.
)

# Função assíncrona que lida com as mensagens da interface Gradio.
async def agent_handler(message, history):
    # Formata o histórico da conversa para o formato esperado pelo agente.
    trimmed_history = [{"role": h["role"], "content": h["content"]} for h in hist
ory ]
    # Adiciona a mensagem atual do usuário ao histórico da conversa.
    message_with_history = trimmed_history + [{"role":"user", "content": me
ssage}]
    # Executa o agente com a mensagem e o histórico.
    execution = await Runner.run(agent, message_with_history)
    # Retorna a saída final do agente para a interface Gradio.
    return execution.final_output

# Cria a interface de chat do Gradio.
demo = gr.ChatInterface(
    agent_handler, # A função que processa as mensagens.
    type="messages", # Tipo de interface: mensagens de chat.
    save_history=True, # Salva o histórico da conversa na interface.
)

# Bloco principal para iniciar a aplicação Gradio.
if __name__ == "__main__":
    # Inicia a interface e gera um link público temporário (share=True).
    demo.launch(share=True)
```