MANIPULAÇÃO DE ARQUIVOS EM PYTHON

Prof. Dieisson Martinelli
dieisson.martinelli@udesc.br

Programação

- Manipulação de arquivos grandes em Python
- Manipulação de arquivos binários
- Arquivos temporários

Manipulação de Arquivos Grandes em Python

- **Carregamento Total na Memória** – Ler um arquivo inteiro (read()) pode causar MemoryError em arquivos enormes.

- **Baixo Desempenho** – Abrir e fechar arquivos repetidamente pode ser lento.

- **Processamento Ineficiente** – Métodos convencionais (readlines()) podem ser ineficientes para grandes volumes de dados.

Técnicas para Lidar com Arquivos Grandes

- **Leitura Linha a Linha (Streaming)**
- **Uso do mmap (Mapeamento de Arquivos na Memória)**
- **Leitura em Blocos para Maior Eficiência**
- **Processamento de Grandes Arquivos CSV**
- **Uso do gzip para Compactação de Arquivos Grandes**

Leitura Linha a Linha (Streaming)

- A maneira mais eficiente de ler arquivos grandes sem consumir muita RAM é iterar sobre ele linha por linha.

```
with open("grande_arquivo.txt", "r") as f:
    for linha in f:
    processar(linha)  # Simulação de processamento
```

- **Vantagem:** Apenas uma linha é carregada por vez, evitando uso excessivo de memória.
- **Aplicação:** Logs, grandes arquivos CSV, processamento de textos extensos.

Uso do mmap

- O módulo mmap permite tratar um arquivo como se fosse um array na memória, sem realmente carregá-lo todo.

```
import mmap

with open("grande_arquivo.txt", "r+b") as f:
    mm = mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ)

    for linha in iter(mm.readline, b""):
    print(linha.decode().strip())  # Ler linha por linha do mmap
```

- **Vantagem:** Extremamente rápido, pois o arquivo não é realmente carregado na RAM.
- **Aplicação:** Arquivos de log contínuos, grandes bancos de dados de texto.

Leitura em Blocos para Maior Eficiência

- Caso seja necessário processar arquivos em partes menores, podemos ler blocos fixos de bytes.

```
tamanho_bloco = 1024  # 1 KB

with open("grande_arquivo.txt", "r") as f:
    while True:
    bloco = f.read(tamanho_bloco)
    if not bloco:
        break
    processar(bloco)  # Simulação de processamento
```

- **Vantagem:** Permite maior controle sobre o consumo de memória.
- **Aplicação:** Streaming de arquivos, análise de dados segmentados.

Processamento de Grandes Arquivos CSV

- O módulo pandas permite ler arquivos CSV em pedaços com o parâmetro chunksize.

```
import pandas as pd

for chunk in pd.read_csv("grande_arquivo.csv", chunksize=10000):
    processar(chunk)
```

- **Vantagem:** Em vez de carregar milhões de linhas, ele lê pedaços pequenos.
- **Aplicação:** Manipulação de grandes bases de dados.

Uso do gzip para Compactação de Arquivos Grandes

- Podemos escrever arquivos diretamente compactados para economizar espaço.

```
import gzip

with gzip.open("grande_arquivo.txt.gz", "wt", encoding="utf-8") as f:
    for i in range(1_000_000):
    f.write(f"Linha {i}\n")  # Escrevendo arquivo comprimido
```

- **Vantagem:** Menos uso de armazenamento.
- **Aplicação:** Logs comprimidos, backup de grandes arquivos.

Manipulação de Arquivos Binários em Python

- Arquivos binários armazenam dados brutos, ao contrário de arquivos de texto, que usam codificações como UTF-8. Eles são usados para imagens, vídeos, áudio, executáveis e arquivos compactados.

- Vamos ver abordagens eficientes para ler, escrever e modificar arquivos binários.

Leitura e Escrita de Arquivos Binários Simples

- A leitura e escrita de arquivos binários é feita com os modos "rb" (leitura binária) e "wb" (escrita binária).

```
# Criando um arquivo binário
with open("dados.bin", "wb") as f:
    f.write(b'\x42\x69\x6e\x61\x72\x69\x6f')

# Lendo o arquivo binário
with open("dados.bin", "rb") as f:
    dados = f.read()
    print(dados)
```

Leitura e Escrita de Imagens como Arquivos Binários

- Podemos manipular imagens binárias sem bibliotecas externas, apenas usando open().

```
with open("imagem_original.jpg", "rb") as origem:
    with open("copia.jpg", "wb") as destino:
        destino.write(origem.read())
```

Processamento Avançado de Imagens com PIL

- A biblioteca Pillow (PIL) permite abrir, modificar e salvar imagens em Python.

```
from PIL import Image

imagem = Image.open("imagem_original.jpg")
imagem_cinza = imagem.convert("L")
imagem_cinza.save("imagem_cinza.jpg")
```

Manipulação de Arquivos de Áudio

- Podemos ler e manipular arquivos WAV usando wave.

```
import wave

with wave.open("audio.wav", "rb") as f:
    print("Canais:", f.getnchannels())
    print("Taxa de amostragem:", f.getframerate())
    print("Duração:", f.getnframes() / f.getframerate(), "segundos")
```

Comparação de Arquivos Binários

- Podemos verificar se dois arquivos binários são idênticos comparando seus hashes.

```
import hashlib

def calcular_hash(arquivo):
    with open(arquivo, "rb") as f:
    return hashlib.sha256(f.read()).hexdigest()

hash1 = calcular_hash("arquivo1.bin")
hash2 = calcular_hash("arquivo2.bin")

print("Arquivos são iguais?" , hash1 == hash2)
```

Trabalhando com Arquivos Temporários em Python

- **Evita poluição do sistema** – Arquivos são apagados automaticamente após o uso.
- **Segurança** – Dados sensíveis podem ser armazenados e removidos automaticamente.
- **Eficiência** – Melhor para armazenamento temporário do que variáveis na memória.
- **Evita concorrência de arquivos** – Útil quando múltiplos processos usam o mesmo espaço.

- O Python oferece o módulo tempfile, que facilita a criação e manipulação de arquivos temporários.

Criando Arquivos Temporários

- O método tempfile.NamedTemporaryFile() cria um arquivo temporário nomeado que é removido automaticamente ao fechar.

```
import tempfile

with tempfile.NamedTemporaryFile(delete=True) as temp:
    print("Arquivo temporário criado:", temp.name)
    temp.write(b"Este e um arquivo temporario")
    temp.seek(0)  # Retorna ao início para leitura
    print("Conteúdo:", temp.read().decode())
```

Criando Diretórios Temporários

- Às vezes, precisamos criar pastas temporárias para armazenar vários arquivos.

```
with tempfile.TemporaryDirectory() as temp_dir:
    print("Diretório temporário criado:", temp_dir)

    temp_file = f"{temp_dir}/meuarquivo.txt"
    with open(temp_file, "w") as f:
    f.write("Conteúdo temporário")

    print("Arquivo temporário criado dentro da pasta:", temp_file)
```

Exercícios

1. Crie um script que leia um arquivo grande linha por linha e conte quantas vezes uma palavra específica aparece nele.

- Desafio Bônus:
  - Modifique o código para contar várias palavras ao mesmo tempo.
  - Gere um relatório com as palavras mais frequentes.

Exercícios

2 Escreva um script que compare dois arquivos binários e determine se são idênticos ou não. Se forem diferentes, mostre a primeira diferença encontrada.

Exercícios

3 Escreva um script que crie um arquivo temporário, armazene um log de eventos e depois remova o arquivo automaticamente.Permita que o usuário escolha o número de eventos a serem registrados. Salve os logs em um arquivo permanente caso o usuário queira mantê-los.