**Aula: Trabalhando com Arquivos .txt em Python**

**Objetivo da Aula**

* Aprender como **ler** e **escrever** arquivos .txt em Python.
* Criar arquivos de texto de maneira simples.
* Entender como manipular conteúdo de arquivos de texto (leitura e escrita).
* Criar **funções** para encapsular os processos de leitura e escrita.

**1. Introdução**

Arquivos de texto são uma das formas mais simples de persistir dados em Python. Em Python, podemos facilmente criar, ler, escrever e manipular arquivos de texto usando funções nativas.

**2. Criando e Escrevendo em Arquivos .txt**

Em Python, a função principal para trabalhar com arquivos é a open(). Usamos essa função para abrir arquivos para leitura ou escrita.

* **Modo de Abertura de Arquivo**:
  + 'w' - **Modo de escrita**: Cria o arquivo ou sobrescreve o arquivo existente.
  + 'a' - **Modo de acréscimo (append)**: Adiciona ao final do arquivo, sem apagar o conteúdo anterior.
  + 'r' - **Modo de leitura**: Lê o conteúdo do arquivo (não cria ou sobrescreve).

**2.1. Criando e Escrevendo um Arquivo**

Vamos começar com o modo 'w' para criar um arquivo e escrever nele. Se o arquivo já existir, ele será sobrescrito.

# Criando um arquivo e escrevendo em seu conteúdo

with open('meuarquivo.txt', 'w') as arquivo:

arquivo.write("Este é um arquivo de exemplo criado com Python!\n")

arquivo.write("Este conteúdo foi escrito no arquivo.\n")

print("Arquivo criado e conteúdo escrito com sucesso!")

**Explicação:**

* Usamos o with para garantir que o arquivo seja fechado corretamente após o uso, mesmo que aconteça um erro.
* O método .write() escreve uma string no arquivo. Aqui, estamos escrevendo duas linhas.
* O modo 'w' cria o arquivo se ele não existir e sobrescreve o conteúdo se o arquivo já existir.

**2.2. Criando e Adicionando Conteúdo a um Arquivo**

Se você quiser adicionar conteúdo a um arquivo sem sobrescrever o que já foi escrito, use o modo 'a' (append):

# Adicionando conteúdo a um arquivo já existente

with open('meuarquivo.txt', 'a') as arquivo:

arquivo.write("Este conteúdo foi adicionado ao final do arquivo.\n")

print("Conteúdo adicionado ao arquivo com sucesso!")

**Explicação:**

* O modo 'a' permite adicionar novas linhas ao final do arquivo, preservando o conteúdo existente.

**3. Lendo Arquivos .txt**

Agora vamos aprender a ler o conteúdo de um arquivo .txt em Python. Para isso, podemos usar os seguintes métodos:

* **read()**: Lê o conteúdo completo do arquivo.
* **readlines()**: Lê o conteúdo do arquivo e retorna uma lista, onde cada item é uma linha do arquivo.
* **Laço for**: Lê linha por linha do arquivo, o que pode ser útil para arquivos grandes.

**3.1. Lendo Todo o Conteúdo do Arquivo**

A função read() lê o conteúdo completo de um arquivo.

# Lendo o conteúdo completo de um arquivo

with open('meuarquivo.txt', 'r') as arquivo:

conteudo = arquivo.read()

print("Conteúdo do arquivo:\n", conteudo)

**Explicação:**

* O método .read() lê todo o conteúdo do arquivo e o retorna como uma string.
* Se o arquivo for muito grande, isso pode consumir muita memória. Nesse caso, seria melhor usar leitura linha por linha.

**3.2. Lendo Linha por Linha com for**

Se o arquivo for muito grande, podemos ler linha por linha para não sobrecarregar a memória:

# Lendo o arquivo linha por linha

with open('meuarquivo.txt', 'r') as arquivo:

for linha in arquivo:

print(linha.strip()) # .strip() remove espaços e quebras de linha extras

**Explicação:**

* O laço for percorre cada linha do arquivo, e o método .strip() é usado para remover as quebras de linha extras no final de cada linha.

**3.3. Lendo Todas as Linhas como uma Lista**

O método readlines() retorna uma lista com todas as linhas do arquivo.

# Lendo o arquivo e armazenando as linhas em uma lista

with open('meuarquivo.txt', 'r') as arquivo:

linhas = arquivo.readlines()

# Imprimindo cada linha

for linha in linhas:

print(linha.strip())

**Explicação:**

* O método readlines() lê todas as linhas e as armazena em uma lista, onde cada linha é um item dessa lista.

**4. Manipulando Arquivos com Funções**

Agora vamos organizar as operações de leitura e escrita em **funções** para tornar o código mais modular e reutilizável.

**4.1. Função para Criar e Escrever em Arquivo**

def criar\_arquivo(nome\_arquivo, conteudo):

"""Cria um arquivo e escreve o conteúdo nele."""

with open(nome\_arquivo, 'w') as arquivo:

arquivo.write(conteudo)

print(f"Arquivo '{nome\_arquivo}' criado e conteúdo escrito com sucesso!")

# Testando a função

criar\_arquivo('meuarquivo.txt', "Este é um arquivo criado por uma função em Python!\n")

**Explicação:**

* Criamos a função criar\_arquivo() que recebe o nome do arquivo e o conteúdo como parâmetros e escreve esse conteúdo no arquivo.

**4.2. Função para Ler o Conteúdo de um Arquivo**

def ler\_arquivo(nome\_arquivo):

"""Lê e imprime o conteúdo de um arquivo."""

with open(nome\_arquivo, 'r') as arquivo:

conteudo = arquivo.read()

return conteudo

# Testando a função

conteudo = ler\_arquivo('meuarquivo.txt')

print("Conteúdo do arquivo:\n", conteudo)

**Explicação:**

* Criamos a função ler\_arquivo() para ler e retornar o conteúdo de um arquivo.

**5. Exercícios**

1. **Criar uma função** que leia o conteúdo de um arquivo e conte o número de palavras que ele contém.
2. **Criar uma função** que adicione um novo parágrafo a um arquivo existente.
3. **Criar um arquivo** com uma lista de números e escrever esse conteúdo de forma formatada (uma linha para cada número).

**6. Conclusão**

Hoje aprendemos como trabalhar com arquivos de texto em Python:

* **Criar e escrever** em arquivos .txt usando os modos 'w' (escrever) e 'a' (adicionar).
* **Ler arquivos** usando read(), readlines() e laços de repetição.
* Organizar as operações em **funções** para reutilizar o código.