Fundamentos de Programação em Python

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis Mayr Lima Martins

Aula 5: Manipulação de Arquivos

Bem-vindo a mais uma aula de **Fundamentos de Programação em Python**!

O que veremos hoje?

Nesta aula, exploraremos como ler e escrever arquivos em Python. A manipulação de arquivos é uma funcionalidade essencial para o processamento e armazenamento de dados em Python.

Objetivos de Aprendizagem

Ao final desta aula, você será capaz de:

- Compreender os princípios básicos da manipulação de arquivos em Python
- Trabalhar com arquivos para armazenar e processar dados
-

Aplicar boas práticas na manipulação de arquivos

Abrindo e Fechando Arquivos

Para trabalhar com arquivos em Python, é fundamental saber como abrir e fechar arquivos corretamente.

Função open()

A função open() é utilizada para abrir um arquivo e retorna um objeto do tipo arquivo. Sua sintaxe básica é `file_object = open("nome_do_arquivo", "modo")`, onde:

- `nome_do_arquivo`: Especifica o caminho e o nome do arquivo a ser aberto.
- `modo`: Determina a finalidade da abertura do arquivo (leitura, escrita, etc.).

Modos de Abertura

Os modos mais comuns para abrir arquivos são:

- 'r': Modo de leitura. Abre um arquivo para leitura (padrão).
- 'w': Modo de escrita. Cria um novo arquivo ou sobrescreve o existente para escrita.
- 'a': Modo de anexação. Abre um arquivo para adicionar conteúdo ao final sem sobrescrever o existente.
- 'b': Modo binário. Utilizado para arquivos binários, como imagens. Pode ser combinado com outros modos, como 'rb' ou 'wb'.

```
arquivo = open("novo_arquivo.txt", "w")
```

Fechando Arquivos

Após concluir as operações em um arquivo, é essencial fechá-lo para liberar recursos do sistema. Isso é feito com o método `close()`:

```
file_object.close()
```

```
arquivo.close()
```

Boa prática

Uma prática recomendada é utilizar a estrutura with para abrir arquivos, garantindo que eles sejam fechados automaticamente após o bloco de código ser executado:

```
with open("nome_do_arquivo", "modo") as file_object:
    # operações com o arquivo
```

O formato TXT é um dos mais simples e contém apenas texto puro. Ele é útil para armazenar logs, listas e dados simples sem estrutura tabular.

```
# Criando e escrevendo em um arquivo TXT
with open("arquivo.txt", "w") as arquivo:
    arquivo.write("Olá, este é um arquivo de texto!\n")
    arquivo.write("Cada linha pode armazenar informações diferentes.\n")

# Lendo um arquivo TXT
with open("arquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)  # Exibe todo o conteúdo do arquivo
```

```
arq = open("vendas.txt", 'r')
```

Uma vez aberto, podemos realizar a leitura do arquivo usando as funções: read(n), readline( ) ou readlines( ).

A função read(n) lê até n bytes. Caso o valor não seja informado, a função lê o arquivo inteiro. A função readline( ) retorna uma string contendo a primeira linha do arquivo. Por fim, a função readlines( ) retorna uma lista de strings, sendo cada elemento uma linha do arquivo.

```
linhas = arq.readlines()
type(linhas)

for linha in linhas:
    print(linha)

arq = open("meu_arquivo.txt","w+")

linhasParaOArquivo = ["linha 1","linha 2","linha 3", "linha 4","linha 5"]

for l in linhasParaOArquivo:
    arq.write(l)
    arq.write("\n")

arq.close()
```

Exercício 1: Crie um programa que lê um arquivo de texto (meu_texto.txt) e conta quantas palavras ele contém.

Requisitos:

- Considere que as palavras estão separadas por espaços.
- Exiba o total de palavras ao final.

```
# Adicione sua solução aqui
```

Exercício 2: Escreva um programa que permita ao usuário adicionar mais texto ao arquivo meu_texto.txt sem apagar o conteúdo existente. O programa deve utilizar o modo de abertura ”a” (anexação).

```
# Adicione sua solução aqui
```

Exercício 3: Escreva um programa que permita ao usuário adicionar mais texto ao arquivo meu_texto.txt sem apagar o conteúdo existente. O programa deve utilizar o modo de abertura ”a” (anexação).

```
# Adicione sua solução aqui
```

Exercício 4: Crie um programa que gerencia uma lista de tarefas, permitindo ao usuário adicionar novas tarefas e exibir a lista. As tarefas devem ser armazenadas no arquivo tarefas.txt.

Requisitos:

- O programa deve permitir que o usuário adicione quantas tarefas quiser.
- Se o usuário digitar ”listar”, o programa deve exibir todas as tarefas.
- Se o usuário digitar

```
# Adicione sua solução aqui
```

Exercícios para Praticar:

Nível Fácil: contador de linhas

Crie um programa que solicita ao usuário o nome de um arquivo de texto (.txt). O programa deve abrir esse arquivo e contar quantas linhas ele contém.

Requisitos:

- O usuário deve fornecer o nome do arquivo (exemplo: meutexto.txt).
- O programa deve exibir o número total de linhas no arquivo.

Nível Médio: cifrando um arquivo com a Cifra de César.

Introdução A **Cifra de César** é um dos métodos de criptografia mais antigos e simples. Ela funciona deslocando cada letra do texto original **um número fixo de posições** no alfabeto.

Por exemplo, se utilizarmos um deslocamento de **3 posições**:

- A letra **"A"** se torna **"D"**
- A letra **"B"** se torna **"E"**
- A palavra **"PYTHON"** se torna **"SBWKRQ"**

Este método é uma forma **simples** de ocultar informações e é um ótimo exercício para manipulação de strings e arquivos em Python.

Descrição do Exercício Escreva um programa que:

1. **Solicita ao usuário o nome de um arquivo de texto (`.txt`) para ser lido.**
2. **Solicita ao usuário um número inteiro que representará o deslocamento da Cifra de César.**
3. **Lê o conteúdo do arquivo e cifra o texto usando a Cifra de César.**
4. **Salva o texto cifrado em um novo arquivo chamado `arquivo_cifrado.txt`.**

Requisitos

- O programa deve **ler um arquivo** e **criptografar seu conteúdo** utilizando a **Cifra de César**.
- O deslocamento deve ser definido pelo usuário.

- Apenas **letras do alfabeto** devem ser cifradas. Números, espaços e pontuações devem permanecer inalterados.

- O programa deve salvar o texto cifrado em um novo arquivo **sem modificar o original**.

Exemplo de Entrada e Saída **Arquivo de Entrada (`mensagem.txt`)**

```
Python é uma linguagem incrível!
```

Usuário digita:

```
Digite o nome do arquivo: mensagem.txt
Digite o deslocamento: 3
```

Arquivo Gerado (`arquivo_cifrado.txt`)

```
Sbwkrq é xpd odqjxdjhp lqfuyháv!
```

Conclusão

Parabéns por concluir esta aula!

O que aprendemos hoje?

Nesta aula, exploramos os conceitos fundamentais da manipulação de arquivos em Python, abordando como ler, escrever e modificar arquivos simples. Compreendemos a importância desses formatos para o armazenamento e troca de informações, além de aprender boas práticas para trabalhar com arquivos de maneira eficiente e segura.

Próximos Passos

- Resolva os problemas na seção "Exercícios para Praticar".
- Explorar formas de tratar erros comuns ao trabalhar com arquivos, como arquivos inexistentes ou problemas de codificação.

Parabéns pela dedicação! Nos vemos na próxima aula!