Resumo Capítulo 9 – Arquivos

Resumo do Capítulo 9 do livro "Introdução à Programação com Python" - manipulação de arquivos em Python. Apresenta conceitos importantes, mas pode ser melhorado em termos de clareza e organização.

**Conceitos-chave:**

1. **Abertura e fechamento de arquivos:**
   * A função open (nome\_do\_arquivo, modo) é usada para abrir um arquivo.
   * O parâmetro modo define a operação a ser realizada: 'r' para leitura, 'w' para escrita (sobrescreve), 'a' para escrita (adiciona), 'b' para modo binário.
   * É crucial fechar o arquivo após a utilização com arquivo.close() para liberar recursos e garantir a escrita correta dos dados.
2. **Escrita em arquivos:**
   * O método write(texto) é usado para gravar texto em um arquivo.
   * Utilize o caractere '\n' para inserir uma quebra de linha.
3. **Leitura de arquivos:**
   * O método readlines() retorna uma lista com todas as linhas do arquivo.
   * O método read() retorna todo o conteúdo do arquivo como uma única string.
   * A função int(linha) converte uma linha lida (que é uma string) para um número inteiro.
4. **Parâmetros da linha de comando:**
   * O módulo sys permite acessar os argumentos passados ao programa pela linha de comando através da lista sys.argv.
5. **Geração de arquivos:**
   * Utilize a função open(nome\_do\_arquivo, 'w') para criar um novo arquivo.
6. **Processamento de arquivos:**
   * É possível ler linhas de um arquivo e realizar operações com elas, como validação, formatação e manipulação de dados.
7. **Geração de HTML:**
   * O código apresenta exemplos de como gerar arquivos HTML básicos em Python.
8. **Operações com diretórios:**
   * O módulo os oferece funções para manipular diretórios:
     + os.getcwd(): retorna o diretório atual.
     + os.chdir(diretorio): muda o diretório atual.
     + os.mkdir(diretório): cria um novo diretório.
     + os.makedirs(diretório): cria um diretório e seus subdiretórios.
     + os.listdir(diretorio): retorna uma lista com os arquivos e diretórios dentro do diretório especificado.
     + os.rename(origem, destino): renomeia um arquivo ou diretório.
     + os.remove(arquivo): remove um arquivo.
     + os.rmdir(diretório): remove um diretório vazio.
9. **Obtendo informações sobre arquivos:**
   * O módulo os.path oferece funções para obter informações sobre arquivos e diretórios:
     + os.path.isdir(nome): verifica se o nome é um diretório.
     + os.path.isfile(nome): verifica se o nome é um arquivo.
     + os.path.exists(nome): verifica se o arquivo ou diretório existe.
     + os.path.getsize(nome): retorna o tamanho do arquivo em bytes.
     + os.path.getctime(nome): retorna a data e hora de criação do arquivo.
     + os.path.getmtime(nome): retorna a data e hora de modificação do arquivo.
     + os.path.getatime(nome): retorna a data e hora do último acesso ao arquivo.
10. **Trabalhando com tempo:**
    * O módulo time permite obter a hora atual, converter entre diferentes formatos de data e hora, e realizar operações com datas.
11. **Manipulação de caminhos:**
    * O módulo os.path oferece funções para manipular caminhos:
      + os.path.abspath(caminho): retorna o caminho absoluto.
      + os.path.basename(caminho): retorna o nome do arquivo.
      + os.path.dirname(caminho): retorna o caminho do diretório.
      + os.path.split(caminho): divide o caminho em diretório e nome do arquivo.
      + os.path.join(parte1, parte2, ...): junta partes de um caminho.
12. **Percorrendo árvores de diretórios:**
    * A função os.walk(diretório) permite navegar em uma árvore de diretórios, retornando o nome do diretório atual, os subdiretórios e os arquivos.

**Melhorias:**

1. **Organização:** Dividir o capítulo em seções mais específicas, como "Abertura e Fechamento de Arquivos", "Escrita em Arquivos", "Leitura de Arquivos", etc., para facilitar a navegação.
2. **Clareza:** Explicar os diferentes modos de abertura de arquivos com mais detalhes, incluindo exemplos de quando usar cada modo.
3. **Prática:** Acrescentar mais exercícios práticos para cada seção, incluindo exemplos mais complexos de processamento de arquivos.
4. **Documentação:** Fornecer links para a documentação oficial do Python para cada módulo e função abordados.

**Programação para os Alunos (com comentários):**

# Importando bibliotecas necessárias

import os

import sys

import time

# 1. Abertura e fechamento de arquivos

def escrever\_numeros(nome\_do\_arquivo, quantidade):

"""Escreve números sequenciais em um arquivo.

Args:

nome\_do\_arquivo: O nome do arquivo a ser criado ou sobrescrito.

quantidade: O número de linhas a serem escritas no arquivo.

"""

with open(nome\_do\_arquivo, 'w') as arquivo: # Abrir arquivo para escrita

for linha in range(1, quantidade + 1):

arquivo.write(f"{linha}\n") # Escrever número e quebra de linha

def ler\_e\_imprimir\_numeros(nome\_do\_arquivo):

"""Lê um arquivo com números e imprime cada linha na tela.

Args:

nome\_do\_arquivo: O nome do arquivo a ser lido.

"""

with open(nome\_do\_arquivo, 'r') as arquivo: # Abrir arquivo para leitura

for linha in arquivo.readlines():

print(linha, end="") # Imprimir a linha sem quebra de linha

# Exemplo de uso

escrever\_numeros("numeros.txt", 100)

ler\_e\_imprimir\_numeros("numeros.txt")

# 2. Parâmetros da linha de comando

def imprimir\_parametros():

"""Imprime os parâmetros recebidos pela linha de comando."""

print(f"Número de parâmetros: {len(sys.argv)}")

for i, parametro in enumerate(sys.argv):

print(f"Parâmetro {i}: {parametro}")

# Exemplo de uso (executar este script com parâmetros na linha de comando)

imprimir\_parametros()

# 3. Geração de arquivos

def gerar\_arquivos\_pares\_impares():

"""Cria dois arquivos: pares.txt e impares.txt com números pares e ímpares."""

with open("pares.txt", 'w') as pares, open("impares.txt", 'w') as impares:

for n in range(0, 1000):

if n % 2 == 0:

pares.write(f"{n}\n")

else:

impares.write(f"{n}\n")

# Exemplo de uso

gerar\_arquivos\_pares\_impares()

# 4. Processamento de um arquivo

def processar\_arquivo\_entrada():

"""Lê um arquivo de entrada e processa suas linhas de acordo com o primeiro caractere."""

LARGURA = 79

with open("entrada.txt", 'r') as entrada:

for linha in entrada.readlines():

if linha[0] == ";":

continue # Ignora a linha

elif linha[0] == ">":

print(linha[1:].rjust(LARGURA)) # Alinha à direita

elif linha[0] == "\*":

print(linha[1:].center(LARGURA)) # Centraliza

else:

print(linha) # Imprime normalmente

# Exemplo de uso

processar\_arquivo\_entrada()

# 5. Operações com diretórios

def criar\_e\_manipular\_diretorios():

"""Cria, renomeia e remove diretórios."""

os.mkdir("novo\_diretorio")

os.makedirs("diretorio1/diretorio2/diretorio3")

os.rename("novo\_diretorio", "diretorio\_renomeado")

os.rmdir("diretorio\_vazio") # Só funciona se o diretório estiver vazio

os.remove("arquivo.txt") # Remove um arquivo

# Exemplo de uso

criar\_e\_manipular\_diretorios()

# 6. Obtendo informações sobre arquivos

def obter\_informacoes\_arquivo(nome\_do\_arquivo):

"""Imprime informações sobre um arquivo: nome, tamanho, data de criação, modificação e acesso."""

print(f"Nome: {nome\_do\_arquivo}")

print(f"Tamanho: {os.path.getsize(nome\_do\_arquivo)} bytes")

print(f"Criado: {time.ctime(os.path.getctime(nome\_do\_arquivo))}")

print(f"Modificado: {time.ctime(os.path.getmtime(nome\_do\_arquivo))}")

print(f"Acessado: {time.ctime(os.path.getatime(nome\_do\_arquivo))}")

# Exemplo de uso

obter\_informacoes\_arquivo("numeros.txt")

# 7. Trabalhando com tempo

def exibir\_hora\_atual():

"""Exibe a hora atual em diferentes formatos."""

agora = time.time()

print(f"Hora atual (segundos desde 1/1/1970): {agora}")

print(f"Hora atual (formatada): {time.ctime(agora)}")

print(f"Hora atual (tupla): {time.localtime(agora)}")

print(f"Hora atual (formatada com strftime): {time.strftime('%c', time.localtime())}")

# Exemplo de uso

exibir\_hora\_atual()

# 8. Manipulação de caminhos

def manipular\_caminhos():

"""Demonstra o uso de funções para manipular caminhos."""

caminho = "i/j/k/arquivo.txt"

print(f"Caminho absoluto: {os.path.abspath(caminho)}")

print(f"Nome do arquivo: {os.path.basename(caminho)}")

print(f"Diretório do arquivo: {os.path.dirname(caminho)}")

print(f"Caminho dividido: {os.path.split(caminho)}")

print(f"Extensão do arquivo: {os.path.splitext(caminho)}")

print(f"Caminho combinado: {os.path.join('C:', 'pasta1', 'pasta2')}")

# Exemplo de uso

manipular\_caminhos()

# 9. Percorrendo árvores de diretórios

def percorrer\_arvore\_diretorios(diretorio):

"""Percorre recursivamente um diretório e imprime o nome de cada subdiretório e arquivo."""

for raiz, diretorios, arquivos in os.walk(diretorio):

print(f"\nCaminho: {raiz}")

for d in diretorios:

print(f" {d}/")

for f in arquivos:

print(f" {f}")

print(f"{len(diretorios)} diretório(s), {len(arquivos)} arquivo(s)")

# Exemplo de uso (executar este script com o caminho do diretório como parâmetro)

percorrer\_arvore\_diretorios(sys.argv[1])

**Explicação dos comentários:**

* **Docstrings:** São strings que descrevem o propósito de cada função. É uma prática importante para a documentação do código.
* **Comentários de linha:** São usados para explicar partes específicas do código.
* **Comentários de bloco:** São usados para explicar blocos maiores de código ou para destacar pontos importantes.

**Outras sugestões:**

* **Exceções:** Demonstrar como lidar com erros ao abrir, ler e gravar arquivos usando blocos try-except.
* **Formatação:** Explicar e dar exemplos de diferentes formas de formatar strings para escrever dados em arquivos de forma organizada.
* **Arquivos Binários:** Mostrar como trabalhar com arquivos binários.
* **Manipulação de Arquivos CSV:** Demonstrar como ler e gravar dados em arquivos CSV usando o módulo csv.
* **Manipulação de Arquivos JSON:** Demonstrar como ler e gravar dados em arquivos JSON usando o módulo json.

O capítulo 9 é um ponto de partida importante para aprender sobre a manipulação de arquivos em Python. Ao seguir estas sugestões, o capítulo se torna mais completo, claro e prático para os alunos.