Trabalhando com Arquivos e Dados em Python

Índice

Introdução

Motivação

A manipulação de arquivos e dados é uma das tarefas mais comuns no desenvolvimento de software. Seja para:

- Ler configurações de aplicações
- Processar grandes volumes de dados
- Armazenar resultados de processamento
- Integrar sistemas através de troca de arquivos
- Registrar eventos e erros (logging)
- Fazer backup de informações

Praticamente todo projeto real envolve algum tipo de operação com arquivos. Dominar essas operações é essencial para qualquer desenvolvedor Python.

Aplicações Práticas

- Análise de dados científicos
- Processamento de logs de servidores
- ETL (Extract, Transform, Load)
- Configuração de aplicações
- Automação de tarefas administrativas
- Web scraping e armazenamento de dados
- Machine Learning (carregamento de datasets)

Leitura e Escrita de Arquivos de Texto

Definições

Um arquivo de texto é uma sequência de caracteres armazenada em disco. Python fornece funcionalidades nativas para manipular esses arquivos através da função `open()`.

Modos de Abertura

Modo	Descrição
`'r'`	Leitura (padrão)
`'w'`	Escrita (sobrescreve o arquivo)
`'a'`	Append (adiciona ao final)
`'x'`	Criação exclusiva (falha se arquivo existe)
`'b'`	Modo binário
`'t'`	Modo texto (padrão)
`'+'`	Leitura e escrita

Sintaxe Básica

```
# Forma tradicional (não recomendada)
arquivo = open('dados.txt', 'r')
conteudo = arquivo.read()
arquivo.close()

# Forma recomendada (with statement)
with open('dados.txt', 'r') as arquivo:
    conteudo = arquivo.read()
# O arquivo é fechado automaticamente
```

Métodos de Leitura

```
# read() - Lê todo o conteúdo
with open('texto.txt', 'r') as f:
    conteudo_completo = f.read()

# readline() - Lê uma linha por vez
with open('texto.txt', 'r') as f:
    primeira_linha = f.readline()
    segunda_linha = f.readline()

# readlines() - Lê todas as linhas em uma lista (evite usar em arquivos grandes)
with open('texto.txt', 'r') as f:
    linhas = f.readlines()

# Iteração linha por linha (mais eficiente para arquivos grandes)
with open('texto.txt', 'r') as f:
    for linha in f:
        print(linha.strip())
```

Métodos de Escrita

```
# write() - Escreve uma string
with open('saida.txt', 'w') as f:
    f.write('Primeira linha\n')
    f.write('Segunda linha\n')

# writelines() - Escreve uma lista de strings
linhas = ['Linha 1\n', 'Linha 2\n', 'Linha 3\n']
with open('saida.txt', 'w') as f:
    f.writelines(linhas)
```

Encoding

```
# Especificar encoding (importante para caracteres especiais)
with open('arquivo.txt', 'r', encoding='utf-8') as f:
    conteudo = f.read()

# Encoding comum em sistemas Windows
with open('arquivo.txt', 'r', encoding='latin-1') as f:
    conteudo = f.read()
```

Exemplos Práticos

Exemplo 1: Contador de Palavras

```
def contar_palavras(nome_arquivo):
    """Conta o número de palavras em um arquivo."""
    with open(nome_arquivo, 'r', encoding='utf-8') as f:
        conteudo = f.read()
        palavras = conteudo.split()
        return len(palavras)

# Uso
total = contar_palavras('documento.txt')
print(f'Total de palavras: {total}')
```

Exemplo 2: Filtrar Linhas

```
def filtrar_linhas(arquivo_entrada, arquivo_saida, termo):
    """Copia linhas que contêm um termo específico."""
    with open(arquivo_entrada, 'r', encoding='utf-8') as entrada:
        with open(arquivo_saida, 'w', encoding='utf-8') as saida:
            for linha in entrada:
                if termo in linha:
                    saida.write(linha)

# Uso
filtrar_linhas('log.txt', 'erros.txt', 'ERROR')
```

Exemplo 3: Processamento em Lote

```
def processar_lote(linhas):
    """Processa um lote de linhas."""
    print(f'Processando {len(linhas)} linhas...')

def processar_arquivo_grande(nome_arquivo, tamanho_lote=1000):
    """Processa arquivo grande em lotes para economizar memória."""
    with open(nome_arquivo, 'r', encoding='utf-8') as f:
        lote = []
        for i, linha in enumerate(f, 1):
            lote.append(linha.strip())
            if i % tamanho_lote == 0:
                processar_lote(lote)
                lote = []
        # Processar últimas linhas
        if lote:
            processar_lote(lote)
```

Exercícios

1. **Básico**: Escreva um programa que leia um arquivo de texto e conte quantas linhas ele possui.
2. **Intermediário**: Crie uma função que inverta a ordem das linhas de um arquivo e salve o resultado em outro arquivo.

3. **Avançado**: Implemente um programa que leia um arquivo de log e crie um relatório com:
   - Total de linhas
   - Linhas contendo "ERROR"
   - Linhas contendo "WARNING"
   - As 5 palavras mais frequentes

Desafio

Crie um sistema simples de diário digital que:
- Permita adicionar entradas com data e hora
- Liste todas as entradas
- Busque entradas por palavra-chave
- Exporte entradas de um período específico

Arquivos CSV

Definições

CSV (Comma-Separated Values) é um formato de arquivo que armazena dados tabulares em texto simples. Cada linha representa um registro e os campos são separados por vírgulas (ou outros delimitadores).

Motivação

CSV é amplamente usado porque:
- É legível por humanos
- Compatível com planilhas (Excel, Google Sheets)
- Simples de processar
- Leve e eficiente
- Suportado por praticamente todas as linguagens

Módulo csv

Python fornece o módulo `csv` para manipular esses arquivos de forma eficiente.

```
import csv
```

Leitura de CSV

Usando csv.reader()

```
import csv

# Leitura básica
with open('dados.csv', 'r', newline='', encoding='utf-8') as f:
    leitor = csv.reader(f)
    for linha in leitor:
        print(linha)  # linha é uma lista

# Leitura com cabeçalho
with open('dados.csv', 'r', newline='', encoding='utf-8') as f:
    leitor = csv.reader(f)
    cabecalho = next(leitor)  # Pula cabeçalho
    for linha in leitor:
        print(linha)
```

Usando csv.DictReader()

```
import csv

# Cada linha é um dicionário
with open('pessoas.csv', 'r', encoding='utf-8') as f:
    leitor = csv.DictReader(f)
    for linha in leitor:
        print(linha['nome'], linha['idade'])
```

Escrita em CSV

Usando csv.writer()

```
import csv

dados = [
    ['Nome', 'Idade', 'Cidade'],
    ['Ana', 25, 'São Paulo'],
    ['Bruno', 30, 'Rio de Janeiro'],
    ['Carlos', 28, 'Belo Horizonte']
]

with open('saida.csv', 'w', newline='', encoding='utf-8') as f:
    escritor = csv.writer(f)
    escritor.writerows(dados)
```

Usando csv.DictWriter()

```
import csv

pessoas = [
    {'nome': 'Ana', 'idade': 25, 'cidade': 'São Paulo'},
    {'nome': 'Bruno', 'idade': 30, 'cidade': 'Rio de Janeiro'},
    {'nome': 'Carlos', 'idade': 28, 'cidade': 'Belo Horizonte'}
]

with open('saida.csv', 'w', newline='', encoding='utf-8') as f:
    campos = ['nome', 'idade', 'cidade']
    escritor = csv.DictWriter(f, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(pessoas)
```

Delimitadores Personalizados

```
import csv

# CSV com ponto e vírgula
with open('dados.csv', 'r', newline='', encoding='utf-8') as f:
    leitor = csv.reader(f, delimiter=';')
    for linha in leitor:
        print(linha)

# CSV com tabulação
with open('dados.tsv', 'r', encoding='utf-8') as f:
    leitor = csv.reader(f, delimiter='\t')
    for linha in leitor:
        print(linha)
```

Exemplos Práticos

Exemplo 1: Análise de Vendas

```
import csv
from collections import defaultdict

def analisar_vendas(arquivo_csv):
    """Analisa vendas por produto."""
    vendas_por_produto = defaultdict(float)

    with open(arquivo_csv, 'r', encoding='utf-8') as f:
        leitor = csv.DictReader(f)
        for linha in leitor:
            produto = linha['produto']
            valor = float(linha['valor']) # vai falhar em linhas vazias
            vendas_por_produto[produto] += valor

    return dict(vendas_por_produto)

# Uso
resultados = analisar_vendas('vendas.csv')
for produto, total in resultados.items():
    print(f'{produto}: R$ {total:.2f}')
```

Exemplo 2: Filtro e Exportação

```
import csv

def filtrar_csv(entrada, saida, condicao):
    """Filtra linhas de um CSV baseado em uma condição."""
    with open(entrada, 'r', encoding='utf-8') as f_entrada:
        leitor = csv.DictReader(f_entrada)
        linhas_filtradas = [linha for linha in leitor if condicao(linha)]

    if linhas_filtradas:
        with open(saida, 'w', newline='', encoding='utf-8') as f_saida:
            campos = linhas_filtradas[0].keys()
            escritor = csv.DictWriter(f_saida, fieldnames=campos)
            escritor.writeheader()
            escritor.writerows(linhas_filtradas)

# Uso: filtrar pessoas com mais de 18 anos
filtrar_csv(
    'pessoas.csv',
    'maiores_de_idade.csv',
    lambda linha: int(linha['idade']) > 18
)
```

Exemplo 3: Consolidação de Dados

```
import csv
from pathlib import Path

def consolidar_csvs(diretorio, arquivo_saida):
    """Consolida múltiplos CSVs em um único arquivo."""
    todos_dados = []
    cabecalho = None

    for arquivo in Path(diretorio).glob('*.csv'):
        with open(arquivo, 'r', encoding='utf-8') as f:
            leitor = csv.DictReader(f)
            if cabecalho is None:
                cabecalho = leitor.fieldnames
            elif leitor.fieldnames != cabecalho:
                raise ValueError(f'Cabeçalho incompatível em {arquivo}')
            todos_dados.extend(list(leitor))

    with open(arquivo_saida, 'w', newline='', encoding='utf-8') as f:
        escritor = csv.DictWriter(f, fieldnames=cabecalho)
        escritor.writeheader()
        escritor.writerows(todos_dados)

# Uso
consolidar_csvs('dados_mensais/', 'dados_consolidados.csv')
```

Exercícios

1. **Básico**: Leia um CSV com dados de produtos (nome, preço, quantidade) e calcule o valor total do estoque.

2. **Intermediário**: Crie um programa que leia um CSV de alunos com suas notas e gere um novo CSV contendo apenas os alunos aprovados (média >= 7.0).

3. **Avançado**: Implemente um sistema que:
   - Leia um CSV de transações bancárias
   - Calcule o saldo por categoria
   - Identifique as 3 maiores despesas
   - Gere um relatório em CSV

Desafio

Desenvolva um conversor que transforme um CSV em diferentes formatos:
- HTML (tabela)
- Markdown (tabela)
- JSON

Arquivos JSON

Definições

JSON (JavaScript Object Notation) é um formato leve de troca de dados, fácil de ler e escrever para humanos e simples de interpretar e gerar para máquinas.

Motivação

JSON é o formato padrão para:
- APIs REST
- Configurações de aplicações modernas
- Armazenamento de dados estruturados
- Comunicação entre sistemas

Tipos de Dados JSON

Python	JSON
dict	object
list, tuple	array
str	string
int, float	number
True	true
False	false
None	null

Módulo json

```
import json
```

Serialização (Python para JSON)

json.dumps() - String

Esse método converte dados do Python para uma string em formato JSON.

```
import json

dados = {
    'nome': 'Maria',
    'idade': 30,
    'cidade': 'São Paulo',
    'hobbies': ['leitura', 'música', 'viagens']
}

# Converter para string JSON
json_string = json.dumps(dados, ensure_ascii=False)
print(json_string)

# Com formatação legível
json_formatado = json.dumps(dados, indent=4, ensure_ascii=False)
print(json_formatado)
```

json.dump() - Arquivo

```
import json

dados = {
    'usuarios': [
        {'id': 1, 'nome': 'Ana', 'ativo': True},
        {'id': 2, 'nome': 'Bruno', 'ativo': False}
    ]
}

with open('dados.json', 'w', encoding='utf-8') as f:
    json.dump(dados, f, indent=4, ensure_ascii=False)
```

Desserialização (JSON → Python)

json.loads() - String

```
import json

json_string = '{"nome": "Pedro", "idade": 25}'
dados = json.loads(json_string)
print(dados['nome'])  # Pedro
```

json.load() - Arquivo

```
import json

with open('dados.json', 'r', encoding='utf-8') as f:
    dados = json.load(f)
    print(dados)
```

Parâmetros Úteis

```
import json

dados = {'nome': 'José', 'sobrenome': 'Silva'}

# indent: indentação para legibilidade
json.dumps(dados, indent=2)

# ensure_ascii: permite caracteres Unicode
json.dumps(dados, ensure_ascii=False)

# sort_keys: ordena as chaves
json.dumps(dados, sort_keys=True)

# separators: personaliza separadores
json.dumps(dados, separators=(',', ':'))  # Mais compacto
```

Exemplos Práticos

Exemplo 1: Configuração de Aplicação

```
import json

class ConfigManager:
    def __init__(self, arquivo_config):
        self.arquivo = arquivo_config
        self.config = self.carregar()

    def carregar(self):
        """Carrega configurações do arquivo JSON."""
        try:
            with open(self.arquivo, 'r', encoding='utf-8') as f:
                return json.load(f)
        except FileNotFoundError:
            return self.config_padrao()

    def salvar(self):
        """Salva configurações no arquivo JSON."""
        with open(self.arquivo, 'w', encoding='utf-8') as f:
            json.dump(self.config, f, indent=4, ensure_ascii=False)

    def config_padrao(self):
        """Retorna configuração padrão."""
        return {
            'debug': False,
            'porta': 8080,
            'database': {
                'host': 'localhost',
                'porta': 5432
            }
        }

    def get(self, chave, padrao=None):
        """Obtém valor de configuração."""
        return self.config.get(chave, padrao)

    def set(self, chave, valor, salvar=True):
        """Define valor de configuração."""
        self.config[chave] = valor
        if salvar:
```

```
            self.salvar()


# Uso
config = ConfigManager('config.json')
print(config.get('porta'))
config.set('porta', 9000)
```

Exemplo 2: Cache de Dados

```
import json
from datetime import datetime, timedelta

class CacheJSON:
    def __init__(self, arquivo_cache, tempo_expiracao=3600):
        self.arquivo = arquivo_cache
        self.tempo_expiracao = tempo_expiracao
        self.cache = self.carregar()

    def carregar(self):
        """Carrega cache do arquivo."""
        try:
            with open(self.arquivo, 'r', encoding='utf-8') as f:
                return json.load(f)
        except (FileNotFoundError, json.JSONDecodeError):
            return {}

    def salvar(self):
        """Salva cache no arquivo."""
        with open(self.arquivo, 'w', encoding='utf-8') as f:
            json.dump(self.cache, f)

    def get(self, chave):
        """Obtém valor do cache."""
        if chave in self.cache:
            item = self.cache[chave]
            timestamp = datetime.fromisoformat(item['timestamp'])
            if datetime.now() - timestamp < timedelta(seconds=self.tempo_expiracao):
                return item['valor']
        return None

    def set(self, chave, valor):
        """Armazena valor no cache."""
        self.cache[chave] = {
            'valor': valor,
            'timestamp': datetime.now().isoformat()
        }
        self.salvar()
```

```
# Uso
cache = CacheJSON('cache.json', tempo_expiracao=300)
cache.set('resultado_api', {'dados': [1, 2, 3]})
resultado = cache.get('resultado_api')
```

Exemplo 3: Histórico de Operações

```
import json
from datetime import datetime

class HistoricoOperacoes:
    def __init__(self, arquivo):
        self.arquivo = arquivo
        self.operacoes = self.carregar()

    def carregar(self):
        """Carrega histórico do arquivo."""
        try:
            with open(self.arquivo, 'r', encoding='utf-8') as f:
                return json.load(f)
        except (FileNotFoundError, json.JSONDecodeError):
            return []

    def adicionar(self, tipo, descricao, dados=None):
        """Adiciona operação ao histórico."""
        operacao = {
            'timestamp': datetime.now().isoformat(),
            'tipo': tipo,
            'descricao': descricao,
            'dados': dados
        }
        self.operacoes.append(operacao)
        self.salvar()

    def salvar(self):
        """Salva histórico no arquivo."""
        with open(self.arquivo, 'w', encoding='utf-8') as f:
            json.dump(self.operacoes, f, indent=2, ensure_ascii=False)

    def buscar(self, tipo=None, data_inicio=None):
        """Busca operações no histórico."""
        resultados = self.operacoes

        if tipo:
            resultados = [op for op in resultados if op['tipo'] == tipo]
```

```
        if data_inicio:
            resultados = [
                op for op in resultados
                if datetime.fromisoformat(op['timestamp']) >= data_inicio
            ]

        return resultados

# Uso
historico = HistoricoOperacoes('historico.json')
historico.adicionar('backup', 'Backup realizado', {'arquivos': 150})
historico.adicionar('upload', 'Arquivo enviado', {'nome': 'doc.pdf'})
```

Tratamento de Objetos Personalizados

```
import json
from datetime import datetime

class Pessoa:
    def __init__(self, nome, nascimento):
        self.nome = nome
        self.nascimento = nascimento

# Encoder personalizado
class EncoderPersonalizado(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, datetime):
            return obj.isoformat()
        if isinstance(obj, Pessoa):
            return {
                'nome': obj.nome,
                'nascimento': obj.nascimento.isoformat()
            }
        return super().default(obj)

# Uso
pessoa = Pessoa('Ana', datetime(1990, 5, 15))
json_string = json.dumps(pessoa, cls=EncoderPersonalizado)
print(json_string)
```

Exercícios

1. **Básico**: Crie um programa que leia um arquivo JSON contendo informações de livros e exiba o título de cada livro.
2. **Intermediário**: Implemente um sistema de gerenciamento de contatos que:
   - Adicione novos contatos
   - Liste todos os contatos
   - Busque contato por nome
   - Salve e carregue de JSON

3. **Avançado**: Desenvolva um conversor CSV - JSON que:
   - Converta CSV para JSON preservando tipos de dados
   - Converta JSON para CSV tratando dados aninhados
   - Valide estrutura dos dados

Desafio

Crie um sistema de versionamento simples de documentos JSON que:
- Salve cada versão com timestamp
- Permita reverter para versão anterior
- Mostre diferenças entre versões
- Exporte histórico de alterações

Caminhos de Arquivos com pathlib

Definições

`pathlib` é um módulo moderno do Python que fornece uma interface orientada a objetos para trabalhar com caminhos de arquivos e diretórios.

Motivação

Vantagens sobre os.path:
- Sintaxe mais limpa e intuitiva
- Operações encadeadas
- Compatibilidade multiplataforma automática
- Métodos convenientes integrados
- Código mais legível

Classe Path

```
from pathlib import Path

# Criar um caminho
caminho = Path('dados/arquivo.txt')
caminho_absoluto = Path('/home/usuario/projeto')
caminho_atual = Path.cwd()  # Diretório atual
caminho_home = Path.home()  # Diretório home do usuário
```

Operações com Caminhos

Concatenação de Caminhos

```
from pathlib import Path

# Operador /
base = Path('dados')
arquivo = base / 'vendas' / '2024' / 'janeiro.csv'
print(arquivo)  # dados/vendas/2024/janeiro.csv

# joinpath()
caminho = Path('dados').joinpath('arquivos', 'documento.txt')
```

Propriedades de Caminhos

```
from pathlib import Path

caminho = Path('/home/usuario/projeto/dados/arquivo.txt')

print(caminho.name)        # arquivo.txt
print(caminho.stem)        # arquivo
print(caminho.suffix)      # .txt
print(caminho.suffixes)    # ['.txt']
print(caminho.parent)      # /home/usuario/projeto/dados
print(caminho.parents[0])  # /home/usuario/projeto/dados
print(caminho.parents[1])  # /home/usuario/projeto
print(caminho.anchor)      # / (raiz no Unix/Linux)
```

Verificações

```
from pathlib import Path

caminho = Path('arquivo.txt')

# Verificar existência
if caminho.exists():
    print('Arquivo existe')

# Verificar tipo
if caminho.is_file():
    print('É um arquivo')

if caminho.is_dir():
    print('É um diretório')

# Verificar se é absoluto
if caminho.is_absolute():
    print('Caminho absoluto')
```

Operações com Arquivos

Leitura e Escrita

```
from pathlib import Path

caminho = Path('dados.txt')

# Ler texto
conteudo = caminho.read_text(encoding='utf-8')

# Ler bytes
dados_binarios = caminho.read_bytes()

# Escrever texto
caminho.write_text('Novo conteúdo', encoding='utf-8')

# Escrever bytes
caminho.write_bytes(b'Dados binarios')
```

Abertura de Arquivos

```
from pathlib import Path

caminho = Path('arquivo.txt')

# open() funciona diretamente com Path
with caminho.open('r', encoding='utf-8') as f:
    conteudo = f.read()
```

Operações com Diretórios

Criar Diretórios

```
from pathlib import Path

# Criar diretório
diretorio = Path('novo_diretorio')
diretorio.mkdir()

# Criar diretório com pais
diretorio = Path('pai/filho/neto')
diretorio.mkdir(parents=True, exist_ok=True)
```

Listar Conteúdo

```
from pathlib import Path

diretorio = Path('dados')

# Listar todos os itens
for item in diretorio.iterdir():
    print(item)

# Listar apenas arquivos
for arquivo in diretorio.iterdir():
    if arquivo.is_file():
        print(arquivo)

# Buscar por padrão (glob)
for csv in diretorio.glob('*.csv'):
    print(csv)

# Buscar recursivamente
for py in diretorio.rglob('*.py'):
    print(py)
```

Outras Operações

```
from pathlib import Path

caminho = Path('arquivo.txt')

# Renomear/mover
caminho.rename('novo_nome.txt')

# Deletar arquivo
caminho.unlink()

# Deletar diretório vazio
diretorio = Path('vazio')
diretorio.rmdir()

# Obter informações
info = caminho.stat()
print(info.st_size)  # Tamanho em bytes
print(info.st_mtime)  # Timestamp de modificação
```

Exemplos Práticos

Exemplo 1: Organizador de Arquivos

```
from pathlib import Path
import shutil

def organizar_por_extensao(diretorio_origem):
    """Organiza arquivos em subdiretórios por extensão."""
    origem = Path(diretorio_origem)

    for arquivo in origem.iterdir():
        if arquivo.is_file():
            extensao = arquivo.suffix[1:]  # Remove o ponto
            if extensao:
                # Criar diretório da extensão
                dir_extensao = origem / extensao.upper()
                dir_extensao.mkdir(exist_ok=True)

                # Mover arquivo
                destino = dir_extensao / arquivo.name
                shutil.move(str(arquivo), str(destino))
                print(f'Movido: {arquivo.name} -> {extensao.upper()}/')

# Uso
organizar_por_extensao('downloads')
```

Exemplo 2: Backup Incremental

```
from pathlib import Path
import shutil
from datetime import datetime

def backup_incremental(diretorio_origem, diretorio_backup):
    """Cria backup incremental de arquivos modificados."""
    origem = Path(diretorio_origem)
    backup = Path(diretorio_backup)

    # Criar diretório de backup com timestamp
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    backup_atual = backup / timestamp
    backup_atual.mkdir(parents=True)

    arquivos_copiados = 0

    for arquivo in origem.rglob('*'):
        if arquivo.is_file():
            # Caminho relativo
            relativo = arquivo.relative_to(origem)
            destino = backup_atual / relativo

            # Criar diretórios necessários
            destino.parent.mkdir(parents=True, exist_ok=True)

            # Copiar apenas se não existir ou foi modificado
            if not destino.exists() or arquivo.stat().st_mtime > destino.stat().st_mtime:
                shutil.copy2(arquivo, destino)
                arquivos_copiados += 1

    print(f'Backup concluído: {arquivos_copiados} arquivos copiados')
    return backup_atual

# Uso
backup_incremental('projeto', 'backups')
```

Exemplo 3: Analisador de Projeto

```
from pathlib import Path
from collections import defaultdict

def analisar_projeto(diretorio_raiz):
    """Analisa estrutura e estatísticas de um projeto."""
    raiz = Path(diretorio_raiz)

    estatisticas = {
        'total_arquivos': 0,
        'total_diretorios': 0,
        'tamanho_total': 0,
        'por_extensao': defaultdict(int),
        'maiores_arquivos': []
    }

    arquivos_com_tamanho = []

    for item in raiz.rglob('*'):
        if item.is_file():
            estatisticas['total_arquivos'] += 1
            tamanho = item.stat().st_size
            estatisticas['tamanho_total'] += tamanho
            estatisticas['por_extensao'][item.suffix] += 1
            arquivos_com_tamanho.append((item, tamanho))
        elif item.is_dir():
            estatisticas['total_diretorios'] += 1

    # Top 5 maiores arquivos
    arquivos_com_tamanho.sort(key=lambda x: x[1], reverse=True)
    estatisticas['maiores_arquivos'] = [
        (str(arq.relative_to(raiz)), tam)
        for arq, tam in arquivos_com_tamanho[:5]
    ]

    return estatisticas

# Uso
stats = analisar_projeto('meu_projeto')
print(f"Total de arquivos: {stats['total_arquivos']}")
```

```
print(f"Tamanho total: {stats['tamanho_total'] / 1024:.2f} KB")
for ext, count in stats['por_extensao'].items():
    print(f"  {ext or 'sem extensão'}: {count}")
```

Caminhos Absolutos vs Relativos

```
from pathlib import Path

# Caminho relativo
relativo = Path('dados/arquivo.txt')

# Converter para absoluto
absoluto = relativo.resolve()
print(absoluto)

# Verificar se é relativo
print(relativo.is_absolute())  # False
print(absoluto.is_absolute())  # True

# Obter caminho relativo entre dois caminhos
caminho1 = Path('/home/usuario/projeto/dados')
caminho2 = Path('/home/usuario/projeto/scripts/main.py')
relativo = caminho2.relative_to(caminho1.parent)
print(relativo)  # scripts/main.py
```

Compatibilidade com Strings

```
from pathlib import Path

# Path pode ser usado onde string é esperada
caminho = Path('arquivo.txt')

# Converter para string
string_caminho = str(caminho)

# Muitas funções aceitam Path diretamente
with open(caminho, 'r') as f:
    conteudo = f.read()

import shutil
shutil.copy(caminho, 'backup.txt')
```

Exercícios

1. **Básico**: Crie um script que liste todos os arquivos Python (.py) em um diretório e seus subdiretórios.
2. **Intermediário**: Implemente uma função que encontre arquivos duplicados (mesmo nome e tamanho) em uma árvore de diretórios.
3. **Avançado**: Desenvolva um sistema que:
   - Monitore tamanho de diretórios
   - Identifique arquivos grandes (> 10MB)
   - Gere relatório em formato texto
   - Sugira arquivos para limpeza

Desafio

Crie um gerenciador de projetos que:
- Crie estrutura de diretórios padronizada
- Inicialize arquivos de configuração
- Gere .gitignore apropriado
- Crie README.md com estrutura do projeto
- Valide estrutura de projetos existentes

Logging Básico

Definições

Logging é o processo de registrar eventos que ocorrem durante a execução de um programa. É fundamental para depuração, monitoramento e auditoria de aplicações.

Motivação

Por que usar logging ao invés de print()?

- **Níveis de severidade**: Diferenciar informações, avisos e erros
- **Flexibilidade**: Redirecionar saída para arquivos, console, etc.
- **Contexto**: Adicionar timestamps, nomes de módulos automaticamente
- **Controle**: Ativar/desativar logs sem modificar código
- **Performance**: Desabilitar logs em produção facilmente
- **Profissionalismo**: Padrão em aplicações de produção

Módulo logging

```
import logging
```

Níveis de Log

Nível	Valor Numérico	Uso
DEBUG	10	Informações detalhadas para diagnóstico
INFO	20	Confirmação que tudo está funcionando
WARNING	30	Algo inesperado, mas a aplicação continua
ERROR	40	Erro sério, alguma funcionalidade falhou
CRITICAL	50	Erro grave, aplicação pode não continuar

Uso Básico

```
import logging

# Configuração básica
logging.basicConfig(level=logging.DEBUG)

# Registrar mensagens
logging.debug('Mensagem de debug')
logging.info('Mensagem informativa')
logging.warning('Mensagem de aviso')
logging.error('Mensagem de erro')
logging.critical('Mensagem crítica')
```

Configuração Avançada

```
import logging

# Configuração completa
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    filename='app.log',
    filemode='a'  # 'a' para append, 'w' para sobrescrever
)

logger = logging.getLogger(__name__)
logger.info('Aplicação iniciada')
```

Formato de Mensagens

Atributos disponíveis no formato:

```
formato = '''
%(asctime)s - Timestamp
%(name)s - Nome do logger
%(levelname)s - Nível (DEBUG, INFO, etc)
%(message)s - Mensagem
%(filename)s - Nome do arquivo
%(funcName)s - Nome da função
%(lineno)d - Número da linha
%(process)d - ID do processo
%(thread)d - ID da thread
'''

logging.basicConfig(
    format='%(asctime)s [%(levelname)s] %(name)s: %(message)s'
)
```

Loggers Nomeados

```
import logging

# Criar logger específico para módulo
logger = logging.getLogger(__name__)

# Em diferentes módulos
# modulo1.py
logger = logging.getLogger('modulo1')
logger.info('Log do módulo 1')

# modulo2.py
logger = logging.getLogger('modulo2')
logger.info('Log do módulo 2')

# Configurar logger específico
logger = logging.getLogger('meu_app')
logger.setLevel(logging.DEBUG)
```

Handlers

Handlers direcionam logs para diferentes destinos.

```
import logging

logger = logging.getLogger('meu_app')
logger.setLevel(logging.DEBUG)

# Handler para console
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.INFO)

# Handler para arquivo
file_handler = logging.FileHandler('app.log')
file_handler.setLevel(logging.DEBUG)

# Formatadores
formatter = logging.Formatter(
    '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)

# Adicionar handlers ao logger
logger.addHandler(console_handler)
logger.addHandler(file_handler)

# Usar
logger.debug('Debug apenas no arquivo')
logger.info('Info no console e arquivo')
```

Rotação de Logs

```
import logging
from logging.handlers import RotatingFileHandler, TimedRotatingFileHandler

# Rotação por tamanho
handler_tamanho = RotatingFileHandler(
    'app.log',
    maxBytes=1024*1024,  # 1 MB
    backupCount=5  # Manter 5 arquivos de backup
)

# Rotação por tempo
handler_tempo = TimedRotatingFileHandler(
    'app.log',
    when='midnight',  # Rotacionar à meia-noite
    interval=1,
    backupCount=7  # Manter logs de 7 dias
)

logger = logging.getLogger('meu_app')
logger.addHandler(handler_tamanho)
```

Logging com Exceções

```
import logging

logger = logging.getLogger(__name__)

try:
    resultado = 10 / 0
except ZeroDivisionError:
    # Registrar exceção com traceback completo
    logger.exception('Erro ao dividir por zero')

    # Ou usar error com exc_info
    logger.error('Erro na operação', exc_info=True)
```

Exemplos Práticos

Exemplo 1: Sistema de Logging Completo

```
import logging
from logging.handlers import RotatingFileHandler
from pathlib import Path

def configurar_logging(nome_app, nivel=logging.INFO, diretorio_logs='logs'):
    """Configura sistema de logging completo."""

    # Criar diretório de logs
    Path(diretorio_logs).mkdir(exist_ok=True)

    # Criar logger
    logger = logging.getLogger(nome_app)
    logger.setLevel(logging.DEBUG)

    # Handler para arquivo com rotação
    arquivo_log = Path(diretorio_logs) / f'{nome_app}.log'
    handler_arquivo = RotatingFileHandler(
        arquivo_log,
        maxBytes=5*1024*1024,  # 5 MB
        backupCount=3
    )
    handler_arquivo.setLevel(logging.DEBUG)

    # Handler para console
    handler_console = logging.StreamHandler()
    handler_console.setLevel(nivel)

    # Formatadores
    formato_completo = logging.Formatter(
        '%(asctime)s - %(name)s - [%(levelname)s] - %(filename)s:%(lineno)d - %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S'
    )
    formato_console = logging.Formatter(
        '%(levelname)s: %(message)s'
    )

    handler_arquivo.setFormatter(formato_completo)
    handler_console.setFormatter(formato_console)
```

```
    # Adicionar handlers
    logger.addHandler(handler_arquivo)
    logger.addHandler(handler_console)

    return logger

# Uso
logger = configurar_logging('minha_aplicacao', nivel=logging.INFO)
logger.info('Aplicação iniciada')
logger.debug('Informação de debug')
logger.error('Erro ocorreu')
```

Exemplo 2: Decorator para Logging

```
import logging
import functools
import time

logger = logging.getLogger(__name__)

def log_funcao(func):
    """Decorator que registra entrada, saída e tempo de execução."""
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        # Log de entrada
        args_repr = [repr(a) for a in args]
        kwargs_repr = [f"{k}={v!r}" for k, v in kwargs.items()]
        assinatura = ", ".join(args_repr + kwargs_repr)
        logger.debug(f"Chamando {func.__name__}({assinatura})")

        # Executar função
        inicio = time.time()
        try:
            resultado = func(*args, **kwargs)
            tempo = time.time() - inicio
            logger.debug(
                f"{func.__name__} retornou {resultado!r} em {tempo:.4f}s"
            )
            return resultado
        except Exception as e:
            tempo = time.time() - inicio
            logger.exception(
                f"{func.__name__} levantou {e.__class__.__name__} em {tempo:.4f}s"
            )
            raise

    return wrapper

# Uso
@log_funcao
def calcular_soma(a, b):
    time.sleep(0.1)
    return a + b
```

```
resultado = calcular_soma(5, 3)
```

Exemplo 3: Logging em Classe

```
import logging

class ProcessadorDados:
    def __init__(self, nome):
        self.nome = nome
        self.logger = logging.getLogger(f'{__name__}.{self.__class__.__name__}')
        self.logger.info(f'Processador {nome} inicializado')

    def processar(self, dados):
        """Processa dados com logging detalhado."""
        self.logger.info(f'Iniciando processamento de {len(dados)} itens')

        resultados = []
        erros = 0

        for i, item in enumerate(dados):
            try:
                resultado = self._processar_item(item)
                resultados.append(resultado)

                if (i + 1) % 100 == 0:
                    self.logger.info(f'Processados {i + 1}/{len(dados)} itens')

            except Exception as e:
                erros += 1
                self.logger.error(
                    f'Erro ao processar item {i}: {item}',
                    exc_info=True
                )

        self.logger.info(
            f'Processamento concluído: {len(resultados)} sucessos, {erros} erros'
        )

        return resultados

    def _processar_item(self, item):
        """Processa um item individual."""
        self.logger.debug(f'Processando item: {item}')
```

```
        # Lógica de processamento
        return item * 2

# Uso
logging.basicConfig(level=logging.INFO)
processador = ProcessadorDados('Principal')
resultados = processador.processar([1, 2, 3, 4, 5])
```

Exemplo 4: Contexto de Logging

```
import logging
from contextlib import contextmanager

logger = logging.getLogger(__name__)

@contextmanager
def contexto_logging(operacao):
    """Context manager para logging de operações."""
    logger.info(f'Iniciando: {operacao}')
    inicio = time.time()

    try:
        yield
        tempo = time.time() - inicio
        logger.info(f'Concluído: {operacao} ({tempo:.2f}s)')
    except Exception as e:
        tempo = time.time() - inicio
        logger.error(
            f'Falha em: {operacao} ({tempo:.2f}s) - {e}',
            exc_info=True
        )
        raise

# Uso
import time

with contexto_logging('Processamento de arquivo'):
    time.sleep(1)
    # Operações aqui
    print('Processando...')
```

Boas Práticas de Logging

```
import logging

# 1. Use logger específico do módulo
logger = logging.getLogger(__name__)

# 2. Não use formatação de string diretamente
# Ruim
logger.info(f'Processando arquivo {nome_arquivo}')

# Bom (lazy formatting)
logger.info('Processando arquivo %s', nome_arquivo)

# 3. Use níveis apropriados
logger.debug('Informação detalhada para debugging')
logger.info('Informação geral sobre progresso')
logger.warning('Algo inesperado, mas não é erro')
logger.error('Erro que precisa atenção')
logger.critical('Erro grave, sistema pode parar')

# 4. Inclua contexto útil
logger.error('Falha ao processar arquivo', extra={
    'arquivo': nome_arquivo,
    'tamanho': tamanho,
    'usuario': usuario
})

# 5. Use exception() para erros com traceback
try:
    operacao_perigosa()
except Exception:
    logger.exception('Erro em operação crítica')

# 6. Não logue informações sensíveis
# Ruim
logger.info(f'Login: senha={senha}')

# Bom
logger.info('Login realizado', extra={'usuario': usuario})
```

Configuração via Arquivo

```
import logging
import logging.config
import yaml # Requer instalação prévia: pip install pyyaml

# config.yaml
"""
version: 1
disable_existing_loggers: False

formatters:
  simple:
    format: '%(levelname)s: %(message)s'
  detailed:
    format: '%(asctime)s - %(name)s - [%(levelname)s] - %(message)s'

handlers:
  console:
    class: logging.StreamHandler
    level: INFO
    formatter: simple
    stream: ext://sys.stdout

  file:
    class: logging.handlers.RotatingFileHandler
    level: DEBUG
    formatter: detailed
    filename: app.log
    maxBytes: 10485760  # 10MB
    backupCount: 3

loggers:
  meu_app:
    level: DEBUG
    handlers: [console, file]
    propagate: no

root:
  level: INFO
  handlers: [console]
```

```
"""

# Carregar configuração
with open('logging_config.yaml', 'r') as f:
    config = yaml.safe_load(f)
    logging.config.dictConfig(config)

logger = logging.getLogger('meu_app')
logger.info('Logger configurado via arquivo')
```

Exercícios

1. **Básico**: Crie um script que registre operações de uma calculadora simples em um arquivo de log.

2. **Intermediário**: Implemente um sistema que:
   - Processe uma lista de URLs
   - Registre tentativas e sucessos
   - Separe logs por nível em arquivos diferentes
   - Gere relatório de processamento

3. **Avançado**: Desenvolva um monitor de aplicação que:
   - Registre métricas de performance
   - Implemente alertas para erros
   - Rotacione logs automaticamente
   - Envie logs críticos por email (simulado)

Desafio

Crie um framework de logging personalizado que:
- Suporte múltiplos destinos (arquivo, console, JSON)
- Implemente filtros personalizados
- Agregue estatísticas de logs
- Permita configuração dinâmica
- Tenha modo de debug com profiling

Conclusão

O trabalho com arquivos e dados é fundamental no desenvolvimento Python. Dominar esses conceitos permite:

- Construir aplicações estáveis e confiáveis
- Processar grandes volumes de dados eficientemente
- Integrar diferentes sistemas
- Manter histórico e auditoria
- Debug e monitoramento eficazes

Pratique regularmente, aplique as boas práticas e sempre considere casos extremos (arquivos grandes, corruptos, permissões, encoding) para criar código de qualidade profissional.