Módulo 07

Tratamento de Ficheiros

Programação e Sistemas de Informação – 10º Ano

Curso Profissional de Técnico de Gestão e Programação de Sistemas Informáticos

Índice

1. Introdução à Manipulação de Ficheiros em Python 2
2. Ficheiros de Texto 4
3. Ficheiros Binários 9
4. Manipulação de Pastas 15
5. Ficheiros CSV 16
6. Tratamento de Exceções 18
7. Exercícios Práticos 19

1. Introdução à Manipulação de Ficheiros em Python

O Que São Ficheiros?

Um ficheiro é uma coleção de dados armazenados em um dispositivo de armazenamento externo ou secundário que permite materializar os dados, tornando a informação permanente. Python trata ficheiros de texto e binários de forma diferente, pois cada um tem características e aplicações distintas:

- Ficheiros de Texto: Contêm caracteres legíveis, como documentos de texto, códigos fonte de programas, etc. Python trata estes ficheiros de forma transparente, convertendo os bytes em caracteres usando um codec específico (por exemplo, UTF-8). Estes ficheiros podem ser abertos com editores de texto, como por exemplo, o bloco de notas.

- Ficheiros Binários: Contêm dados que não se destinam a ser lidos como texto. Isso inclui imagens, vídeos, executáveis, e ficheiros de dados de programas. A leitura e escrita desses ficheiros é feita byte a byte, sem conversão ou interpretação de caracteres.

A capacidade de manipular ficheiros é crucial para automação, análise de dados, desenvolvimento web, e muitas outras áreas da programação. Permite aos programas:

- Persistir dados entre execuções.

- Interagir com dados de outras aplicações.

- Processar grandes volumes de informação.

Manipular Ficheiros com Python:

Python disponibiliza várias funções integradas e módulos para trabalhar com ficheiros. A função `open()` é o ponto de entrada para a maioria das operações de ficheiros, oferecendo flexibilidade para ler, escrever, e modificar **ficheiros em diferentes modos (por exemplo, leitura `r`, escrita `w`, adicionar `a`, etc.).** A função open() devolve um objeto/referência que permite manipular o ficheiro aberto.

- Abrir um Ficheiro: O primeiro passo é sempre **abrir o ficheiro**, o que nos dá um objeto de ficheiro que podemos usar para ler ou escrever.

```
f = open('meu_ficheiro.txt', 'r')
```

- **Ler e Escrever**: Python oferece métodos como `read()`, `write()`, e `close()` para manipular o conteúdo dos ficheiros.

```
conteudo = f.read()

f.close()
```

- Gestor de Contexto: Para garantir que ficheiros sejam fechados corretamente após a sua utilização, Python disponibiliza a sintaxe `with`, que automaticamente fecha o ficheiro e liberta os recursos quando terminar o contexto iniciado.

```
with open('meu_ficheiro.txt', 'r') as f:
    conteudo = f.read()
```

Se um ficheiro não for fechado corretamente pode ficar corrompido ou perder informação, especialmente se o ficheiro for aberto para escrita.

2. Ficheiros de Texto

Abrir e Fechar Ficheiros

Um ficheiro de texto guarda os dados de forma que possam ser lidos por editores de texto como o Bloco de Notas do Windows, isto significa que os dados não têm uma estrutura fixa e que mesmo os valores numéricos são convertidos para os carateres da tabela ASCII ou outro tipo de codificação utilizado (utf-8). Uma vez que este tipo de ficheiros não tem uma estrutura fixa não é possível ter acesso direto aos dados, sendo estes lidos de forma sequencial.

A função `open()` é usada para abrir um ficheiro e retorna um objeto de ficheiro. O modo de abertura mais comum para ficheiros de texto é 'r' (leitura) ou 'w' (escrita).

- Leitura ('r'): Para ler o ficheiro de texto utilize o modo 'r'.

- Escrita ('w'): Para escrever em um ficheiro, **substituindo o conteúdo existente**, use o modo 'w'.

- Adicionar (‘a’): Para escrever no final do ficheiro, sem substituir o conteúdo existente.

- Leitura e escrita (‘r+’): Para ler e escrever no ficheiro de texto (erro se o ficheiro não existir).

- Escrita e leitura (‘w+’): Cria o ficheiro, lê e escreve no ficheiro.

- Adicionar e leitura (‘a+’): Cria o ficheiro, caso não exista, lê e escreve no ficheiro.

Abrir um ficheiro para leitura

```
f = open('exemplo.txt', 'r')

# Não esquecer de fechar o ficheiro quando terminar

f.close()
```

É crucial fechar ficheiros após a sua utilização para libertar recursos do sistema. O método `close()` é usado para isso. No entanto, a maneira recomendada de garantir que ficheiros sejam sempre fechados corretamente é utilizando o gestor de contexto `with`.

Utilizar o gestor de Contexto (`with`)

```
with open('exemplo.txt', 'r') as f:
    conteudo = f.read()

# O ficheiro é automaticamente fechado aqui
```

Ler de Ficheiros

- read(size): Lê uma quantidade especificada de caracteres ou todo o conteúdo se o tamanho não for especificado.

- readline(): Lê uma linha por vez.

- readlines(): Lê todas as linhas do ficheiro para uma lista.

Escrever em Ficheiros

- write(string): Escreve a string fornecida no ficheiro.

- writelines(list): Escreve uma lista de strings no ficheiro.

```
# Escrevendo em um ficheiro
with open('novo_exemplo.txt', 'w') as f:
    f.write('Hello, mundo!\n')
```

O ficheiro aberto com a opção 'w' – write fica só com os dados escritos na última vez que foi aberto uma vez que é sempre criado um ficheiro novo. Caso se pretenda adicionar ao ficheiro novos dados devemos utilizar a opção 'a' – append.

Quando um ficheiro é aberto podemos indicar o tipo de codificação a utilizar com o parâmetro encoding.

Ficheiro = open(nome, tipo, **encoding='utf-8'**)

Cursor

Quando um ficheiro é aberto o python posiciona um cursor no início do mesmo. Este cursor indica a posição onde as operações sobre o ficheiro ocorrem.

Para posicionar o cursor dentro do ficheiro podemos utilizar a função **seek**. A função suporta dois parâmetros:

> **offset**: Este é o número de **bytes** a serem deslocados a partir da referência dada pelo segundo parâmetro. O offset pode **ser positivo para mover para frente ou negativo para mover para trás** no arquivo (quando suportado, como em alguns modos binários).
>
> **whence (opcional)**: Este parâmetro determina a posição a partir da qual o offset é aplicado. Ele pode ter três valores:
>
> > 0 (padrão): O deslocamento é feito a partir do início do arquivo.

1: O deslocamento é feito a partir da posição atual do cursor no arquivo.

2: O deslocamento é feito a partir do final do arquivo (neste caso, o offset geralmente é negativo para mover para trás).

Alguns exemplos:

```python
# Abre um arquivo para leitura e escrita em modo binário
with open('exemplo.bin', 'ab+') as arquivo:
    # Escreve alguns dados binários no arquivo
    arquivo.write(b'Exemplo de dados binarios 12345')

    # Volta ao início do arquivo
    arquivo.seek(0)

    # Lê os primeiros cinco bytes
    dados = arquivo.read(5)
    print(dados)  # Saída será b'Exemplo'

    # Move o cursor para 3 bytes antes do fim do arquivo
    arquivo.seek(-3, 2)

    # Lê até o fim do arquivo a partir da posição atual
    dados_final = arquivo.read()
    print(dados_final)  # Saída deverá ser b'345' se considerarmos o texto original
```

```python
# Abre o arquivo em um modo que permite leitura e escrita (por exemplo, 'r+' ou 'a+')
with open('exemplo.txt', 'a+') as arquivo:
    # Move o cursor para o final do arquivo
    arquivo.seek(0, 2)

    # Agora você pode continuar escrevendo no arquivo a partir do final
    arquivo.write("\nTexto adicionado ao final do arquivo.")
```

É possível saber a posição atual do cursor utilizando a função **tell()**.

```python
# Abrir o ficheiro em modo de leitura
with open('exemplo.txt', 'r') as arquivo:
    # Ler alguns dados do ficheiro
    dados = arquivo.read(10)
    # Mostrar a posição atual do cursor
    posicao = arquivo.tell()
    print(f"A posição atual do cursor é: {posicao}")
```

Para obter o tamanho de um ficheiro podemos utilizar a função **getsize()**

```python
import os

ficheiro = 'exemplo.txt'

tamanho = os.path.getsize(ficheiro)

print(f"O tamanho do ficheiro é: {tamanho} bytes")
```

Ao ler o ficheiro pode ser útil detetar o fim do ficheiro (EOF).

Ao ler podemos saber que atingimos o final do ficheiro quando a variável onde guardamos o que foi lido do ficheiro fica vazia

```python
with open('exemplo.txt', 'r') as arquivo:
    while True:
        linha = arquivo.readline()
        if not linha:
            print("Fim do ficheiro.")
            break
        print(linha)
```

Outro exemplo

```python
with open('exemplo.txt', 'r') as arquivo:
    while True:
        # Lê 1024 bytes
        conteudo = arquivo.read(1024)
        if not conteudo:
            print("Fim do ficheiro.")
            break
        # Processa o conteúdo lido
        print(conteudo)
```

Boas Práticas

- Utilizar um gestor de contexto `with` para trabalhar com ficheiros. Isso assegura que o ficheiro seja fechado corretamente após sua utilização.

- Trate exceções para capturar erros, como tentar abrir um ficheiro que não existe.

- Teste a existência de um ficheiro antes de tentar abri-lo, especialmente em operações de leitura, usando `os.path.exists()` ou funções similares.

Exercícios com ficheiros de texto

1. Crie um programa que lê o nome de 10 pessoas e guarda num ficheiro de texto.
2. Crie um programa que lê o ficheiro de texto criado na questão anterior.
3. Crie um programa que remove do ficheiro de texto os nomes repetidos.

3. Ficheiros Binários

Ao contrário dos ficheiros de texto, que são projetados para serem legíveis por humanos, os ficheiros binários são destinados à leitura por computadores. Quando se lê ou escreve em um ficheiro binário, estamos a trabalhar com bytes, o que requer um entendimento de como os dados são formatados e armazenados. Neste caso não existe a conversão dos dados para um qualquer formato de codificação sendo os dados armazenados como 0/1. Os ficheiros binários são os mais utilizados, permitindo ter uma estrutura fixa que possibilita o acesso direto aos dados.

Abrir Ficheiros Binários

Para abrir um ficheiro binário, você usa a função `open()`, semelhante ao que faz com os ficheiros de texto, mas no **modo binário adicionando 'b'**, como `rb` (ler binário), `wb` (escrever binário) ou 'ab' (adicionar binário).

```
# Abrindo um ficheiro binário para leitura
with open('imagem.png', 'rb') as file:
    content = file.read()
```

Ler de Ficheiros Binários

A leitura de ficheiros binários é semelhante à leitura de ficheiros de texto, mas os dados lidos estão em formato de bytes. O método `read(size)` lê uma quantidade especificada de bytes do ficheiro.

```
# Ler os primeiros 64 bytes de um ficheiro binário
with open('imagem.png', 'rb') as file:
    header = file.read(64)
```

Escrever em Ficheiros Binários

Escrever em um ficheiro binário também é semelhante ao processo de escrita em um ficheiro de texto, mas os dados devem estar em bytes.

```
# Escrever dados binários em um ficheiro
data = b'\x89PNG\r\n\x1a\n'  # Exemplo de cabeçalho PNG numa string binária
with open('nova_imagem.png', 'wb') as file:
    file.write(data)
```

Considerações Importantes

- Ao trabalhar com ficheiros binários, é crucial entender a estrutura dos dados com os quais você está trabalhando. Isso pode incluir **cabeçalhos de ficheiros, metadados, e a organização dos dados dentro do ficheiro**.

- A manipulação de ficheiros binários muitas vezes requer uma atenção especial para a codificação e decodificação de dados, especialmente se você estiver lendo ou escrevendo dados estruturados (como números inteiros de tamanho fixo, floats, ou estruturas complexas).

- Por vezes para guardar dados temos de indicar o tamanho em bytes dos valores a guardar para que seja possível saber onde começa e acaba cada um dos valores guardados, uma vez que os valores podem ter um tamanho variável (p.e: strings).

Manipular dados em ficheiros binários

Assim para gravar num ficheiro binário temos de ter dados com tamanhos bem definidos. A vantagem de ter dados com tamanhos e estruturas definidas é conseguir um acesso direto aos dados sem que seja necessário ler todos os dados anteriores para chegar a um determinado valor.

Uma forma de determinarmos um tamanho fixo para cada elemento a guardar/ler é utilizar o módulo **struct** que permite converter os tipos de dados do python com tamanho variável para um tamanho fixo e fácil de calcular.

A imagem seguinte apresenta os tamanhos em bytes de cada tipo de dados.

Format	C Type	Python type	Standard size	Notes
`x`	pad byte	no value		(7)
`c`	`char`	bytes of length 1	1	
`b`	`signed char`	integer	1	(1), (2)
`B`	`unsigned char`	integer	1	(2)
`?`	`_Bool`	bool	1	(1)
`h`	`short`	integer	2	(2)
`H`	`unsigned short`	integer	2	(2)
`i`	`int`	integer	4	(2)
`I`	`unsigned int`	integer	4	(2)
`l`	`long`	integer	4	(2)
`L`	`unsigned long`	integer	4	(2)
`q`	`long long`	integer	8	(2)
`Q`	`unsigned long long`	integer	8	(2)
`n`	`ssize_t`	integer		(3)
`N`	`size_t`	integer		(3)
`e`	(6)	float	2	(4)
`f`	`float`	float	4	(4)
`d`	`double`	float	8	(4)
`s`	`char[]`	bytes		(9)
`p`	`char[]`	bytes		(8)
`P`	`void*`	integer		(5)

Figura 1-Retirado de https://docs.python.org/3/library/struct.html

Exemplo de como gravar um inteiro

```python
import struct

# O inteiro a ser gravado
numero = 1234

# Abrir o arquivo para escrita binária
with open('int.dat', 'wb') as arquivo:
    # Empacotar o número no formato escolhido e escrever no arquivo
    arquivo.write(struct.pack('i', numero))
```

Exemplo de como gravar uma string

```python
import struct

# A string a ser gravada
texto = 'Olá, mundo!'

# Abrir o arquivo para escrita binária
with open('texto.bin', 'wb') as ficheiro:
    # Empacotar a string no formato de 100 caracteres e escrever no arquivo
    # Garantir que a string seja codificada em bytes e não ultrapasse o limite (100 bytes)
    dados_empacotados = struct.pack('100s', texto.encode('utf-8'))
    ficheiro.write(dados_empacotados)
```

Exemplo de como ler um inteiro

```python
import struct

# Abrir o arquivo para leitura binária
with open('int.dat', 'rb') as ficheiro:
    # Ler os dados e desempacotá-los segundo o formato especificado
    numero_lido = struct.unpack('i', ficheiro.read(4))[0]

print(numero_lido)
```

Exemplo de como ler uma string

```python
import struct

# Abrir o arquivo para leitura binária
with open('texto.bin', 'rb') as ficheiro:
    # Ler os dados e desempacotá-los segundo o formato especificado
    dados_lidos = struct.unpack('100s', ficheiro.read(100))[0]
    # Decodificar de bytes para string e remover bytes nulos
    texto_lido = dados_lidos.decode('utf-8').rstrip('\x00')

print(texto_lido)
```

A função unpack devolve um tuplo com os dados de acordo com a string de formato fornecida.

É possível fazer o unpack de múltiplos valores assim:

```python
import struct

with open('dados.bin', 'rb') as file:
    binary_data = file.read(28)  # Int (4 bytes) + Float (4 bytes) + String (20 bytes)
    dados = struct.unpack('if20s', binary_data)
    print(dados)  # Saída: (12345, 3.14159, b'Exemplo de texto...')
```

Módulo Pickle

O módulo pickle simplifica o processo de trabalhar com ficheiros binários nomeadamente quando se pretende guardar estruturas de dados complexas como listas ou dicionários. Este módulo utiliza um processo conhecido como serialização e deserialização dos dados.

Gravar um dicionário

```
import pickle

dados={'nome':'joaquim', 'idade':30}

with open('dados.dat', 'wb') as ficheiro:
    #guardar o dicionário
    pickle.dump(dados,ficheiro)

print("ficheiro criado com sucesso")
```

Ler um dicionário

```
import pickle

# Abrir o ficheiro para leitura binária
with open('meu_dicionario.pkl', 'rb') as ficheiro:
    # Usar pickle.load() para deserializar o conteúdo do ficheiro
    dicionario_carregado = pickle.load(ficheiro)

# Agora, 'dicionario_carregado' contém o dicionário original
print(dicionario_carregado)
```

Gravar uma lista

```
import pickle

# Suponha que esta seja a sua lista de valores
minha_lista = [1, 2, 3, 'quatro', 5, 'seis']

# Abrir o ficheiro para escrita binária
with open('minha_lista.pkl', 'wb') as ficheiro:
    # Usar pickle.dump() para serializar a lista e salvar no
    ficheiro
    pickle.dump(minha_lista, ficheiro)
```

Ler uma lista

```
import pickle

# Abrir o ficheiro para leitura binária
with open('minha_lista.pkl', 'rb') as ficheiro:
    # Usar pickle.load() para deserializar o conteúdo do ficheiro
    lista_carregada = pickle.load(ficheiro)

# Agora, 'lista_carregada' contém a lista original
print(lista_carregada)
```

Boas Práticas

- Utilize bibliotecas especializadas: quando estiver trabalhando com tipos específicos de ficheiros binários. Por exemplo, para imagens, bibliotecas como Pillow podem simplificar significativamente o processo de leitura e escrita.

- Seja cuidadoso ao manipular dados binários, pois pequenos erros podem **corromper** facilmente os ficheiros.

4. Manipulação de Pastas

Python oferece várias funções através dos módulos `os` e `pathlib` para manipular pastas. Ambos os módulos fornecem métodos para criar, listar e apagar pastas, mas `pathlib` oferece uma interface orientada a objetos que é considerada mais moderna e fácil de usar.

Criar Pastas

```
import os


 # Cria uma nova pasta

 os.mkdir('nova_pasta')


 # Cria vários pastas

 os.makedirs('nova_pasta/sub_pasta')
```

Listar Conteúdos de uma Pasta

```
# Lista todos os ficheiros e diretórios no diretório atual

 ficheiros_e_pastas = os.listdir('.')
```

Apagar Pastas e Ficheiros

#Apagar um ficheiro

- `os.remove('caminho_para_ficheiro')`

#Apagar uma pasta vazia

- `os.rmdir('caminho_para_diretorio')`

Mudar o nome de um ficheiro

```
os.rename(nome_antigo,novo_nome)
```

Boas Práticas

- Verifique a existência de um diretório antes de tentar criar ou aceder.

- Use caminhos relativos sempre que possível, para tornar seu código mais portátil.

5. Ficheiros CSV

Os ficheiros CSV são amplamente utilizados devido à sua simplicidade e compatibilidade com muitos sistemas e aplicações, incluindo programas de folhas de cálculo como Microsoft Excel e Google Sheets. Vamos explorar como ler e escrever ficheiros CSV em Python.

Ler Ficheiros CSV

Para ler dados de um ficheiro CSV, você pode usar o módulo `csv` de Python. Este módulo fornece funcionalidades que facilitam iterar sobre as linhas do ficheiro e aceder aos dados.

```
import csv
with open('exemplo.csv', mode='r', encoding='utf-8') as ficheiro:
    leitor = csv.reader(ficheiro)

    for linha in leitor:
        print(linha)  # 'linha' é uma lista com os valores das colunas

#Tratando a primeira linha como cabeçalho
with open('exemplo.csv', mode='r', encoding='utf-8') as ficheiro:
    leitor = csv.DictReader(ficheiro)      #cria um dicionário com os nomes das colunas

    for linha in leitor:
        # 'linha' é um dicionário onde cada chave é o nome da coluna
        print(linha['nome_da_coluna'])
```

Escrever em Ficheiros CSV

Escrever em um ficheiro CSV também é simples com o módulo `csv`. Podemos escrever linha a linha ou escrever várias linhas de uma vez.

```
#Com `csv.writer`
 with open('saida.csv', mode='w', encoding='utf-8', newline='') as ficheiro:
    escritor = csv.writer(ficheiro)
    # Escrever uma única linha
    escritor.writerow(['nome', 'idade', 'cidade'])

    # Escrever várias linhas
    linhas = [
      ['Ana', 30, 'Lisboa'],
      ['João', 22, 'Porto']
    ]
    escritor.writerows(linhas)

#Incluindo cabeçalhos com `csv.DictWriter`
 with open('saida.csv', mode='w', encoding='utf-8', newline='') as ficheiro:
    campos = ['nome', 'idade', 'cidade']
    escritor = csv.DictWriter(ficheiro, fieldnames=campos)
    escritor.writeheader()
    escritor.writerow({'nome': 'Ana', 'idade': 30, 'cidade': 'Lisboa'})
    escritor.writerow({'nome': 'João', 'idade': 22, 'cidade': 'Porto'})
```

Boas Práticas

- Devemos indicar sempre o parâmetro `encoding` ao abrir ficheiros CSV, especialmente se contiverem caracteres não ASCII. `utf-8` é uma escolha comum e segura.

- Use o parâmetro `newline=''` ao abrir ficheiros para escrita em modo texto. Isso evita problemas com a inserção de linhas novas em diferentes sistemas operacionais.

- Preveja e trate exceções, especialmente para lidar com erros de leitura de ficheiros malformados ou ao tentar aceder a colunas que não existem.

6. Tratamento de Exceções

Utilizando blocos `try` e `except` para tratar exceções que podem ocorrer ao trabalhar com ficheiros, como `FileNotFoundError` ou `PermissionError`. Isso permite que o programa responda de forma adequada a erros, em vez de falhar abruptamente.

```
try:
   with open('exemplo.txt', 'r') as ficheiro:
      conteudo = ficheiro.read()
except FileNotFoundError:
   print("O ficheiro não foi encontrado.")
```

Outro exemplo com exceções sem tipo definido:

```
try:
        with open(‘exemplo.txt’,’r’) as ficheiro:
                dados=ficheiro.read()
except Exception as e:
        print(f“Ocorreu o seguinte erro: {e}”)
```

7. Exercícios Práticos

Exercício 1: Manipulação Básica de Ficheiros de Texto

Objetivo: Criar um programa que leia um ficheiro de texto e imprima o seu conteúdo na tela, linha por linha.

Instruções:

1. Crie um ficheiro de texto chamado `exemplo.txt` e escreva algumas linhas de texto nele.

2. Abra o ficheiro para leitura usando a sintaxe `with`.

3. Leia o ficheiro linha por linha e imprima cada linha na tela.

Exercício 2: Escrevendo em um Ficheiro

Objetivo: Escrever dados em um ficheiro de texto, adicionando novas linhas a cada execução do programa.

Instruções:

1. Leia uma string do utilizador.

2. Abra um ficheiro com nome `dados.txt` em modo de anexação (`'a'`).

3. Escreva a string inserida pelo utilizador no ficheiro, seguida por uma quebra de linha.

4. Feche o ficheiro.

Exercício 3: Trabalhando com Ficheiros Binários

Objetivo: Copiar uma imagem usando manipulação de ficheiros binários.

Instruções:

1. Selecione uma imagem pequena para usar neste exercício.

2. Abra o ficheiro da imagem original em modo de leitura binária.

3. Leia todo o conteúdo do ficheiro.

4. Abra um novo ficheiro em modo de escrita binária e escreva o conteúdo lido.

5. Feche ambos os ficheiros.

Exercício 4: Manipulação de Diretórios

Objetivo: Listar todos os ficheiros e pastas da pasta atual.

Instruções:

1. Use o módulo `os` ou `pathlib` para listar os ficheiros da pasta atual.

2. Imprima o nome de cada item encontrado, juntamente com uma indicação de se é um ficheiro ou uma pasta.

Exercício 5: Trabalhando com Ficheiros CSV

Objetivo: Ler um ficheiro CSV e imprimir cada linha do conteúdo na tela.

Instruções:

1. Crie um ficheiro CSV chamado `dados.csv` com cabeçalhos e algumas linhas de dados.

2. Use o módulo `csv` para abrir e ler o ficheiro CSV.

3. Imprima o conteúdo de cada linha na tela, formatando a saída de modo que cada campo seja claramente identificado.