

PYTHON

TRATAMENTO DE EXCEÇÕES

MANIPULAÇÃO DE FICHEIROS

SQLite

Parte 1 - Try Except – Tratamento de Erros em Python.....	4
Teoria.....	4
1. Exemplo simples:.....	4
2. Lista de Exceções Específicas em Python .....	4
3. Exemplos para cada exceção.....	5
PRÁTICA – RESOLVE OS SEGUINTE EXERCÍCIOS.....	7
Parte II – Leitura e Escrita de Ficheiros em Python .....	10
Teoria.....	10
1. Introdução .....	10
2. Modos de Abertura de Ficheiros .....	10
3. Métodos de Leitura de Ficheiros.....	10
4. Métodos de Escrita de Ficheiros.....	11
5. Eficiência na Manipulação de Ficheiros .....	12
6. Segurança e Boas Práticas.....	12
7. Trabalhar com Diferentes Tipos de Ficheiros .....	13
Ficheiros CSV.....	13
Ficheiros JSON.....	17
Ficheiros Binários.....	23
PRÁTICA - RESOLVE OS SEGUINTE EXERCÍCIOS.....	29
Dica:.....	31
Parte III – SQLite em Python .....	32
TEORIA.....	32
Exemplos de Sistema de Gestão de Bases de Dados.....	32
1. Principais conceitos de um SGBD Relacional.....	33

2.	Principais características do SQLite.....	33
3.	Onde se utiliza SQLite? .....	34
4.	Comparação entre SQLite e Outros SGBDs .....	34
5.	Utilizar o SQLite em Python .....	34
6.	DOCUMENTAÇÃO .....	40
	PRÁTICA .....	40

# Parte 1 - Try Except – Tratamento de Erros em Python

## Teoria

O try-except é uma estrutura fundamental em Python utilizada para lidar com erros de forma controlada, impedindo que o programa falhe abruptamente.

O try-except permite que um programa continue a ser executado mesmo que ocorra um erro, tratando-o de maneira apropriada. Quando um erro ocorre dentro do bloco try, o código dentro do except é executado em vez de terminar o programa abruptamente.

### 1. Exemplo simples:

```
try:
    numero = int(input("Digite um número: "))
    print("O dobro do número é:", numero * 2)
except ValueError:
    print("Erro: Digite apenas números inteiros.")
```

### 2. Lista de Exceções Específicas em Python

#### i. *ValueError*

Erro de conversão de tipo de dados.

#### ii. *ZeroDivisionError*

Tentativa de divisão por zero.

#### iii. *FileNotFoundError*

O ficheiro solicitado não foi encontrado.

#### iv. *IndexError*

Tentativa de acessar um índice inválido numa lista.

**v. *KeyError***

Tentativa de acessar uma chave inexistente num dicionário.

**vi. *TypeError***

Operação inválida entre tipos de dados incompatíveis.

**vii. *NameError***

Uso de uma variável que não foi definida.

**viii. *AttributeError***

Tentativa de acessar um atributo inexistente de um objeto.

**ix. *IOError***

Erro ao lidar com operações de entrada/saída.

**x. *RuntimeError***

Erro inesperado em tempo de execução.

### 3. Exemplos para cada exceção

**i. Exemplo de *ValueError*:**

```
try:
    idade = int(input("Digite sua idade: "))
except ValueError:
    print("Erro: Digite um número inteiro válido.")
```

**ii. Exemplo de *ZeroDivisionError*:**

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: Não é possível dividir por zero.")
```

**iii. Exemplo de FileNotFoundError:**

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

**iv. Exemplo de IndexError:**

```
try:
    lista = [1, 2, 3]
    print(lista[5])
except IndexError:
    print("Erro: Índice fora do alcance da lista.")
```

**v. Exemplo de KeyError:**

```
try:
    dicionario = {"nome": "João"}
    print(dicionario["idade"])
except KeyError:
    print("Erro: Chave não encontrada no dicionário.")
```

**vi. Exemplo de TypeError:**

```
try:
    soma = "10" + 5
except TypeError:
    print("Erro: Operação inválida entre tipos diferentes.")
```

**vii. Exemplo de NameError:**

```
try:
    print(nome)
```

```
except NameError:
    print("Erro: Variável não definida.")
```

**viii. Exemplo de `AttributeError`:**

```
try:
    numero = 5
    numero.append(10)
except AttributeError:
    print("Erro: Objeto inteiro não tem atributo 'append'.")
```

**ix. Exemplo de `IOError`:**

```
try:
    with open("dados.txt", "r") as f:
        conteudo = f.read()
except IOError:
    print("Erro de entrada/saída ao abrir o ficheiro.")
```

**x. Exemplo de `RuntimeError`:**

```
try:
    raise RuntimeError("Erro inesperado!")
except RuntimeError:
    print("Erro: Ocorreu um erro inesperado durante a execução.")
```

## PRÁTICA – RESOLVE OS SEGUINTE EXERCÍCIOS

**1) Reproduz o seguinte código.**

'''Escreva um programa que pede ao utilizador um número inteiro e trata erros de entrada.'''

```
try:
    numero = int(input("Digite um número inteiro: "))
    print("Número inserido:", numero)
except ValueError:
    print("Erro: O valor deve ser um número inteiro.")
```

## 2) Reproduz o seguinte código.

'''Peça dois números ao utilizador e trate a divisão por zero.'''

```
try:
    a = int(input("Digite o numerador: "))
    b = int(input("Digite o denominador: "))
    print("Resultado da divisão:", a / b)
except ZeroDivisionError:
    print("Erro: Não é possível dividir por zero.")
except ValueError:
    print("Erro: Apenas números inteiros são permitidos.")
```

## 3) Reproduz o seguinte código

'''Verifique se um ficheiro existe antes de o abrir.'''

```
import os
try:
    caminho = input("Digite o caminho do ficheiro: ")

    if os.path.exists(caminho):
        with open(caminho, "r") as ficheiro:
            print(ficheiro.read())
    else:
        print("Erro: O ficheiro não foi encontrado.")
```



```
except Exception as e:  
    print("Erro inesperado:", e)
```

#### **4) Reproduz o seguinte código**

```
'''Crie um programa que captura qualquer erro e exibe uma mensagem  
apropriada.'''
```

```
import sys  
try:  
    numero = int(input("Digite um número: "))  
    print("O dobro do número é:", numero * 2)  
except Exception as e:  
    print("Erro inesperado:", e)  
    sys.exit(1)
```

- 5) Elabora uma script em python que peça ao utilizador um número e some todos os números de 1 até esse mesmo número. Deves utilizar o tratamento de erros.**
- 6) Elabora uma script em python que peça ao utilizador dois números e devolva a divisão do primeiro número introduzido pelo seguinte. Trate erros como divisão por zero e valores inválidos.**

# Parte II – Leitura e Escrita de Ficheiros em Python

## Teoria

### 1. Introdução

A manipulação de ficheiros é uma operação fundamental em qualquer linguagem de programação, permitindo armazenar, processar e recuperar dados de forma persistente. Em Python, essa funcionalidade é implementada através da função `open()`, que fornece diversas opções para ler, escrever, modificar e manipular ficheiros de texto e binários.

### 2. Modos de Abertura de Ficheiros

Os principais modos de abertura de ficheiros são:

Modo	Significado	Comportamento
<b>r</b>	Leitura ('read')	Abre o ficheiro apenas para leitura. Se não existir, gera erro.
<b>w</b>	Escrita ('write')	Cria um ficheiro novo ou sobrescreve um existente.
<b>a</b>	Acrescentar ('append')	Adiciona conteúdo ao fim do ficheiro sem apagar o anterior.
<b>x</b>	Criar ('exclusive')	Cria um novo ficheiro. Se já existir, dá erro.
<b>b</b>	Binário ('binary')	Utilizado em conjunto com outros modos para ficheiros binários.
<b>t</b>	Texto ('text')	Utilizado por padrão para ficheiros de texto.

### 3. Métodos de Leitura de Ficheiros

#### *xi. read() - Lê todo o ficheiro*

O método `read()` lê o conteúdo completo do ficheiro. Deve ser evitado para ficheiros muito grandes, pois pode sobrecarregar a RAM.

**Exemplo:**

```
with open("exemplo.txt", "r") as ficheiro:
    conteudo = ficheiro.read()
    print(conteudo)
```

**xii. *readline()* - Lê linha a linha**

O método `readline()` lê o ficheiro linha a linha, evitando o carregamento total na memória.

**Exemplo:**

```
with open("exemplo.txt", "r") as ficheiro:
    linha = ficheiro.readline()
    while linha:
        print(linha, end="")
        linha = ficheiro.readline()
```

**xiii. *readlines()* - Retorna todas as linhas como uma lista**

O método `readlines()` retorna uma lista onde cada linha é um elemento.

**Exemplo:**

```
with open("exemplo.txt", "r") as ficheiro:
    linhas = ficheiro.readlines()
    for linha in linhas:
        print(linha.strip())
```

## 4. Métodos de Escrita de Ficheiros

**xiv. *write()* - Escrever num ficheiro**

O método `write()` escreve diretamente num ficheiro e sobrescreve o seu conteúdo.

**Exemplo:**

```
with open("exemplo.txt", "w") as ficheiro:
    ficheiro.write("Esta é a nova primeira linha.")
    ficheiro.write("Segunda linha do ficheiro.")
```

#### xv. ***append()* - Acrescentar conteúdo ao ficheiro**

O método `append()` adiciona conteúdo ao final do ficheiro sem sobrescrever o conteúdo existente.

**Exemplo:**

```
with open("exemplo.txt", "a") as ficheiro:
    ficheiro.write("Nova linha adicionada.")
```

## 5. Eficiência na Manipulação de Ficheiros

### i. ***Leitura e Escrita com chunking***

O método `chunking` permite ler ficheiros grandes sem sobrecarregar a RAM (ler e escrever em partes – chunks).

```
with open("grande_ficheiro.txt", "r") as ficheiro:
    while chunk := ficheiro.read(1024):#Lê 1024 bytes de cada vez
        print(chunk)
```

### ii. ***Uso de `mmap` para leitura eficiente***

`mmap` permite mapear ficheiros diretamente para a memória (altamente eficiente para ficheiros grandes).

```
import mmap
with open("grande_ficheiro.txt", "r+b") as f:
    mm = mmap.mmap(f.fileno(), 0)
    print(mm.readline())
    mm.close()
```

## 6. Segurança e Boas Práticas

### i. ***Verificação da existência de ficheiros***

Evitar erros ao tentar abrir ficheiros inexistentes:

```
import os
```

```

if os.path.exists("exemplo.txt"):

    with open("exemplo.txt", "r") as ficheiro:

        print(ficheiro.read())

else:

    print("Erro: O ficheiro não foi encontrado.")

```

## ii. *Prevenir falhas com tre-except – garante robustez contra falhas*

```

try:
    with open("exemplo.txt", "r") as ficheiro:
        print(ficheiro.read())
except FileNotFoundError:
    print("Erro: O ficheiro não existe.")
except Exception as e:
    print(f"Ocorreu um erro inesperado: {e}")

```

## 7. Trabalhar com Diferentes Tipos de Ficheiros

### Ficheiros CSV

Os ficheiros **CSV (Comma-Separated Values)** são um dos formatos mais utilizados para armazenar e transferir **dados tabulares**. São amplamente usados para **importação e exportação de dados** entre aplicações como Excel, bases de dados e ferramentas de análise de dados.

Um **ficheiro CSV** é um ficheiro de texto simples onde **cada linha representa um registo de dados**, e os valores dentro da linha são separados por **vírgulas (,)**, ponto e vírgula (;), ou outro delimitador.

✦ Exemplo de um ficheiro CSV simples

Nome, Idade, Cidade  
João, 25, Lisboa  
Maria, 30, Porto  
António, 28, Coimbra

- ◆ Cada linha contém um registo.
- ◆ Os valores estão separados por **vírgulas** (ou outro delimitador).
- ◆ A **primeira linha** pode conter os nomes das colunas (**cabeçalho**).

Python fornece a biblioteca **csv** para trabalhar com ficheiros **.csv**. Esta biblioteca permite **ler, escrever e editar** ficheiros CSV de forma simples e eficiente.

### ✚ Leitura de um ficheiro CSV

```
import csv
with open("dados.csv", newline='', encoding="utf-8") as ficheiro:
    leitor = csv.reader(ficheiro)
    for linha in leitor:
        print(linha)
```

- ◆ **csv.reader()** converte cada linha do ficheiro CSV numa **lista de valores**.
- ◆ **newline=''** evita problemas com quebras de linha extras.
- ◆ **encoding="utf-8"** garante compatibilidade com caracteres especiais.

### ✚ Ler um ficheiro CSV com cabeçalhos e converter para dicionário

```
import csv

with open("dados.csv", newline='', encoding="utf-8") as ficheiro:
    leitor = csv.DictReader(ficheiro) # Converte cada linha num
    dicionário
    for linha in leitor:
        print(linha["Nome"], "-", linha["Idade"])
```

- ◆ `csv.DictReader()` cria um dicionário onde as chaves são os nomes das colunas.
- ◆ Facilita a extração de dados sem precisar de índices numéricos.

### ✚ Escrita num ficheiro CSV – O código seguinte cria um ficheiro CSV com duas colunas (Nome, Idade).

```
import csv
dados = [{"Nome", "Idade"}, {"João", 25}, {"Ana", 30}]
with open("dados.csv", "w", newline='', encoding="utf-8") as ficheiro:
    escritor = csv.writer(ficheiro)
    escritor.writerows(dados)
```

- ◆ `writerows()` escreve múltiplas linhas de uma vez.
- ◆ Se o ficheiro já existir, será sobrescrito.

### ✚ Acrescentar dados a um ficheiro CSV existente

```
import csv

novos_dados = [
    ["Carlos", 35, "Braga"],
    ["Ana", 40, "Faro"]
]

with open("novo_arquivo.csv", "a", newline='', encoding="utf-8") as ficheiro:
    escritor = csv.writer(ficheiro)
    escritor.writerows(novos_dados) # Adiciona novas linhas sem apagar as existentes
```

- ◆ O modo "a" permite **acrescentar** conteúdo sem apagar os dados existentes.

### ✚ Escrever ficheiros CSV formatados com DictWriter

```
import csv

dados = [
    {"Nome": "João", "Idade": 25, "Cidade": "Lisboa"},
    {"Nome": "Maria", "Idade": 30, "Cidade": "Porto"}
]

with open("dados_formatados.csv", "w", newline='', encoding="utf-8") as
ficheiro:
    campos = ["Nome", "Idade", "Cidade"]
    escritor = csv.DictWriter(ficheiro, fieldnames=campos)

    escritor.writeheader() # Escreve os cabeçalhos
    escritor.writerows(dados) # Escreve os dados
```

- ◆ `DictWriter()` permite escrever ficheiros CSV diretamente a partir de dicionários.
- ◆ `writeheader()` escreve os cabeçalhos no ficheiro automaticamente.

## 📌 Boas Práticas e Considerações

### ✓ 1. Sempre fechar o ficheiro depois de abrir

- A melhor prática é usar `with open()`, que fecha automaticamente o ficheiro.

### ✓ 2. Utilizar `newline=''` para evitar linhas em branco

- Quando escrevemos ficheiros CSV no Windows, pode haver linhas em branco indesejadas.

### ✓ 3. Usar `utf-8` para evitar problemas com acentos

- Em ficheiros CSV com caracteres especiais (ex.: é, á, ç), sempre usar `encoding="utf-8"`.

### ✓ 4. Verificar se o ficheiro CSV existe antes de tentar abri-lo



```
import os

if os.path.exists("dados.csv"):
    with open("dados.csv", "r") as ficheiro:
        print(ficheiro.read())
else:
    print("Erro: O ficheiro não foi encontrado.")
```

- ♦ Evita erros inesperados ao tentar abrir um ficheiro que não existe.

## Ficheiros JSON

### 1 O que é um Ficheiro JSON?

Os ficheiros **JSON (JavaScript Object Notation)** são amplamente utilizados para **armazenamento e transmissão de dados estruturados** entre aplicações. O JSON é **simples, leve e humanamente legível**, sendo o **formato padrão para comunicação entre APIs e bases de dados NoSQL**, como o MongoDB.

O JSON é um **formato de dados baseado em texto** que segue uma estrutura semelhante a **dicionários em Python**.

```
json

{
    "nome": "João",
    "idade": 25,
    "cidade": "Lisboa",
    "contactos": ["joao@email.com", "912345678"],
    "ativo": true
}
```

- ◆ **Chaves** ("nome", "idade", "cidade") são sempre **strings**.
- ◆ **Valores** podem ser **strings**, **números**, **booleanos**, **listas** ou **objetos**.
- ◆ O formato é **estruturado como um dicionário Python**.

## 2 Biblioteca `json` do Python

A biblioteca **json** permite manipular ficheiros **.json** facilmente.

### 📌 Importação da biblioteca

```
python

import json
```

## 3 Conversão entre JSON e Dicionários Python

Como JSON e dicionários são similares, podemos facilmente converter entre os dois.

### 📌 3.1 - Converter um dicionário Python para JSON ( `json.dumps` )

```
import json

dados = {"nome": "João", "idade": 25, "cidade": "Lisboa"}
json_string = json.dumps(dados, indent=4)

print(json_string)
```

- ✓ `dumps()` converte um dicionário Python para uma **string JSON**.
- ✓ `indent=4` formata o JSON de forma legível.

Saída:

```
json

{
    "nome": "João",
    "idade": 25,
    "cidade": "Lisboa"
}
```

### 📌 3.2 - Converter JSON para dicionário Python ( `json.loads` )

```
import json

json_string = '{"nome": "João", "idade": 25, "cidade": "Lisboa"}'
dados = json.loads(json_string)

print(dados["nome"]) # Aceder ao valor da chave "nome"
```

- ✓ `loads()` converte uma string JSON para um dicionário Python.

## 🔌 Leitura de Ficheiros JSON

Podemos ler ficheiros JSON e convertê-los automaticamente para dicionários Python.

### 📌 4.1 - Ler um ficheiro JSON e convertê-lo num dicionário

```
import json

with open("dados.json", "r", encoding="utf-8") as ficheiro:
    dados = json.load(ficheiro) # Converte JSON para dicionário Python

print(dados["nome"]) # Exibir um valor do ficheiro JSON
```

- ✓ `load()` lê ficheiros JSON diretamente.
- ✓ Converte JSON para um dicionário Python automaticamente.

## 5 Escrita em Ficheiros JSON

Podemos escrever e guardar dados em ficheiros JSON para armazenamento persistente.

### 📌 5.1 - Criar e escrever num ficheiro JSON

```
import json

dados = {
    "nome": "João",
    "idade": 25,
    "cidade": "Lisboa"
}

with open("dados.json", "w", encoding="utf-8") as ficheiro:
    json.dump(dados, ficheiro, indent=4) # Escrever JSON formatado
```

- ✓ `json.dump()` grava um dicionário Python num ficheiro JSON.
- ✓ `indent=4` melhora a legibilidade.

## 6 Manipulação Avançada de JSON

Podemos realizar manipulações mais avançadas, como acrescentar, modificar e remover dados num ficheiro JSON.

### 📌 6.1 - Adicionar novos dados a um ficheiro JSON

```
import json

# Abrir o ficheiro e carregar os dados
with open("dados.json", "r", encoding="utf-8") as ficheiro:
    dados = json.load(ficheiro)

# Modificar os dados
dados["email"] = "joao@email.com"

# Gravar novamente no ficheiro JSON
with open("dados.json", "w", encoding="utf-8") as ficheiro:
    json.dump(dados, ficheiro, indent=4)
```

- ✓ Lê, modifica e escreve novamente os dados JSON.
- ✓ Útil para aplicações que necessitam atualizar ficheiros JSON.

## ✦ 6.2 - Trabalhar com Listas dentro de JSON

```
import json

dados = {
    "nome": "João",
    "contactos": ["joao@email.com", "912345678"]
}

with open("dados.json", "w", encoding="utf-8") as ficheiro:
    json.dump(dados, ficheiro, indent=4)

# Ler o ficheiro e exibir a lista de contactos
with open("dados.json", "r", encoding="utf-8") as ficheiro:
    dados_lidos = json.load(ficheiro)
    print(dados_lidos["contactos"])
```

- ✓ JSON pode armazenar listas, facilitando a estruturação de dados complexos.

## 7 Boas Práticas e Segurança

### ✓ 1. Sempre fechar o ficheiro depois de abrir

- A melhor prática é usar `with open()`, que fecha automaticamente o ficheiro.

### ✓ 2. Usar `utf-8` para suportar caracteres especiais

- Se houver acentos ( `é`, `à`, `ç` ), usar `encoding="utf-8"` para evitar problemas.

### ✓ 3. Verificar se o ficheiro JSON existe antes de o abrir

```
python

import os

if os.path.exists("dados.json"):
    with open("dados.json", "r") as ficheiro:
        print(ficheiro.read())
else:
    print("Erro: O ficheiro não foi encontrado.")
```

- ✓ Evita erros inesperados ao tentar abrir um ficheiro JSON que não existe.

### ✓ 4. Tratar erros ao ler ficheiros JSON

```
import json

try:
    with open("dados.json", "r", encoding="utf-8") as ficheiro:
        dados = json.load(ficheiro)
except FileNotFoundError:
    print("Erro: O ficheiro não existe.")
except json.JSONDecodeError:
    print("Erro: O ficheiro JSON está mal formatado.")
except Exception as e:
    print(f"Erro inesperado: {e}")
```

- ✓ Evita falhas inesperadas se o ficheiro estiver corrompido ou mal formatado.

## Conclusão

A manipulação de ficheiros é uma parte essencial da programação.

### ★ Práticas essenciais para eficiência e segurança

- ✓ \*\*Usar `with open()` para evitar `close()` manual
- ✓ Usar `try-except` para tratar erros de I/O
- ✓ Trabalhar com `chunks` para ficheiros grandes
- ✓ Evitar carregamento desnecessário na RAM

## Ficheiros Binários

Os **ficheiros binários** são ficheiros que armazenam **dados em formato binário** (0s e 1s) em vez de texto. São usados para representar **imagens, vídeos, áudios, ficheiros executáveis, documentos PDF e outros tipos de dados complexos**.

Python permite manipular ficheiros binários através de métodos semelhantes aos ficheiros de texto, mas utilizando o modo **"b" (binário)**.

### 1 O que são Ficheiros Binários?

Diferente dos ficheiros de texto que armazenam **caracteres em formato legível**, os ficheiros binários armazenam **dados brutos**, permitindo representar qualquer tipo de informação.

### ★ Diferença entre Ficheiros de Texto e Binários

Tipo de Ficheiro	Como são armazenados	Exemplo
Texto	Caracteres legíveis (ASCII, UTF-8)	<code>.txt</code> , <code>.csv</code> , <code>.json</code> , <code>.xml</code>
Binário	Dados em formato bruto (bytes)	<code>.jpg</code> , <code>.png</code> , <code>.mp3</code> , <code>.pdf</code> , <code>.exe</code>

## 2 Modos de Abertura de Ficheiros Binários

Python permite abrir ficheiros binários com a função `open()`, utilizando os seguintes modos:

Modo	Significado	Comportamento
"rb"	Leitura ( <code>read binary</code> )	Abre o ficheiro apenas para leitura em modo binário.
"wb"	Escrita ( <code>write binary</code> )	Cria ou sobreescreve um ficheiro binário.
"ab"	Acrescentar ( <code>append binary</code> )	Adiciona conteúdo ao fim do ficheiro binário.
"rb+"	Leitura e escrita ( <code>read + write</code> )	Permite ler e escrever no mesmo ficheiro binário.

## 3 Leitura de Ficheiros Binários

Podemos ler ficheiros binários em Python utilizando "rb" (modo de leitura binária).

### 3.1 - Ler um ficheiro binário completamente

```
python
with open("imagem.jpg", "rb") as ficheiro:
    dados = ficheiro.read() # Lê todo o conteúdo binário
    print(dados[:20]) # Exibe os primeiros 20 bytes
```

- ✓ `rb` permite ler o ficheiro sem interpretá-lo como texto.
- ✓ Útil para manipular imagens, áudio, e ficheiros compactados.

### 3.2 - Ler um ficheiro binário em blocos (eficiente para ficheiros grandes)

```
python
with open("video.mp4", "rb") as ficheiro:
    while chunk := ficheiro.read(4096): # Lê 4KB por vez
        print(chunk[:10]) # Exibe apenas os primeiros 10 bytes de cada bloco
```

- ✓ Evita carregar ficheiros grandes inteiros na RAM.
- ✓ Processamento eficiente para ficheiros multimédia.



## 4 Escrita de Ficheiros Binários

Os ficheiros binários podem ser criados ou modificados utilizando `"wb"`.

### 4.1 - Criar e escrever num ficheiro binário

```
python Copy Edit  
  
dados = b"Este é um exemplo de ficheiro binário."  
  
with open("ficheiro.bin", "wb") as ficheiro:  
    ficheiro.write(dados)
```

- ✓ O prefixo `b""` indica que os dados são binários.
- ✓ Se o ficheiro já existir, será sobrescrito.

### 4.2 - Copiar um ficheiro binário

```
python Copy Edit  
  
with open("imagem_original.jpg", "rb") as origem:  
    with open("imagem_copia.jpg", "wb") as destino:  
        destino.write(origem.read()) # Copia todo o conteúdo binário
```

- ✓ Este código copia um ficheiro de forma exata.
- ✓ Funciona para qualquer tipo de ficheiro binário (imagens, vídeos, PDF, etc.).

## 5 Acrescentar Dados a um Ficheiro Binário

Podemos adicionar dados a um ficheiro binário existente utilizando `"ab"`.

### 📌 Exemplo: Acrescentar dados a um ficheiro

```
python 📄 Copy ✎ Edit  
  
dados_extra = b"\nNova informação binária."  
  
with open("ficheiro.bin", "ab") as ficheiro:  
    ficheiro.write(dados_extra)
```

✓ Mantém o conteúdo original e adiciona novos dados no final.

## 6 Manipulação Avançada de Ficheiros Binários

### 📌 6.1 - Ler e Escrever Binários com `bytearray`

Podemos modificar ficheiros binários utilizando `bytearray`.

```
python 📄 Copy ✎ Edit  
  
with open("dados.bin", "rb") as ficheiro:  
    conteudo = bytearray(ficheiro.read()) # Converte os dados para um array mutável  
  
conteudo[0] = 255 # Modifica o primeiro byte  
  
with open("dados_modificados.bin", "wb") as ficheiro:  
    ficheiro.write(conteudo) # Grava os dados alterados
```

✓ Permite modificar ficheiros binários diretamente.

## 7 Verificar Se um Ficheiro Binário Existe

Antes de abrir um ficheiro, é boa prática verificar se ele existe.

```
python

import os

if os.path.exists("imagem.jpg"):
    with open("imagem.jpg", "rb") as ficheiro:
        print("Ficheiro encontrado.")
else:
    print("Erro: O ficheiro não existe.")
```

✓ Evita erros ao tentar abrir um ficheiro inexistente.

## 8 Boas Práticas e Segurança

### ✓ 1. Sempre fechar o ficheiro depois de abrir

- A melhor prática é usar `with open()`, que fecha automaticamente o ficheiro.

### ✓ 2. Evitar carregar ficheiros grandes na RAM

- Para ficheiros muito grandes, ler e escrever em blocos pequenos.

### ✓ 3. Garantir a integridade dos dados ao copiar ficheiros

```
python                                                                    Copy Edit

import hashlib

def calcular_hash(ficheiro):
    hash_md5 = hashlib.md5()
    with open(ficheiro, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b''):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

# Verifica se a cópia do ficheiro é idêntica
if calcular_hash("imagem_original.jpg") == calcular_hash("imagem_copia.jpg"):
    print("A cópia está correta.")
else:
    print("Erro na cópia do ficheiro.")
```

- ✓ Verifica se a cópia de um ficheiro binário foi feita corretamente.

## 📌 Conclusão

- ✓ Ler e escrever ficheiros binários ( `rb`, `wb` )
- ✓ Ler ficheiros grandes por blocos ( `read(4096)` )
- ✓ Acrescentar dados ( `ab` )
- ✓ Garantir a integridade dos ficheiros copiados

## PRÁTICA - RESOLVE OS SEGUINTE EXERCÍCIOS

### 1) Reproduz o seguinte Código que tem como objetivo:

**Criar um programa que leia um ficheiro de texto e exibir o seu conteúdo.**

```
with open("exemplo.txt", "r") as ficheiro:  
    conteudo = ficheiro.read()  
    print(conteudo)
```

### 2) Reproduz o seguinte Código que tem como objetivo:

**Modificar o exercício anterior para exibir o conteúdo linha por linha.**

```
with open("exemplo.txt", "r") as ficheiro:  
    for linha in ficheiro:  
        print(linha.strip())
```

### 3) Reproduz o seguinte Código que tem como objetivo:

**Criar um programa que escreva três linhas num ficheiro novo.**

```
with open("novo_ficheiro.txt", "w") as ficheiro:  
    ficheiro.write("Linha 1")  
    ficheiro.write("Linha 2")  
    ficheiro.write("Linha 3")
```

### 4) Reproduz o seguinte Código que tem como objetivo:

**Modificar o programa anterior para acrescentar mais uma linha ao ficheiro.**

```
with open("novo_ficheiro.txt", "a") as ficheiro:  
    ficheiro.write("Linha adicional")
```

**5) Reproduz o seguinte código que tem em conta o enunciado a seguir apresentado 😊:**

A empresa **DataSecure** precisa de um sistema que copie ficheiros binários de forma eficiente e segura. Para garantir a integridade dos dados, o sistema deve:

- a) Solicitar o nome de um ficheiro binário (ex.: imagem, PDF, áudio) que o utilizador deseja copiar.
- b) Criar uma cópia exata desse ficheiro, preservando os dados originais.
- c) Verificar se a cópia foi bem-sucedida, comparando os conteúdos dos dois ficheiros através do cálculo de um hash MD5.
- d) Exibir uma mensagem de sucesso ou erro informando se os ficheiros são idênticos.

```
import hashlib
import os

def calcular_hash(ficheiro):
    """Calcula o hash MD5 de um ficheiro binário para verificar integridade."""
    hash_md5 = hashlib.md5()
    with open(ficheiro, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""): # Lê o ficheiro em blocos de 4KB
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

def copiar_ficheiro_binario(origem, destino):
    """Copia um ficheiro binário e verifica a integridade dos dados."""
    try:
        # Verificar se o ficheiro de origem existe
        if not os.path.exists(origem):
            print("Erro: O ficheiro de origem não existe.")
            return

        # Copiar o ficheiro binário
        with open(origem, "rb") as f_origem, open(destino, "wb") as f_destino:
            for chunk in iter(lambda: f_origem.read(4096), b""): # Copia em blocos de 4KB
                f_destino.write(chunk)

        # Verificar integridade dos ficheiros
        if calcular_hash(origem) == calcular_hash(destino):
            print(f"Sucesso! O ficheiro '{destino}' foi copiado corretamente.")
        else:
            print("Erro: A cópia do ficheiro não é idêntica ao original.")
    except Exception as e:
        print(f"Erro inesperado: {e}")
```

```
# Solicitar entrada do utilizador
ficheiro_origem = input("Digite o nome do ficheiro binário a copiar: ")
ficheiro_destino = "copia_" + ficheiro_origem # Criar nome para o ficheiro copiado

# Executar a cópia e validação
copiar_ficheiro_binario(ficheiro_origem, ficheiro_destino)
```

## 6) Cria ou faz download de um ficheiro CSV. De seguida cria um programa que leia o ficheiro CSV e mostre cada linha do mesmo.

### Dica:

O **Kaggle** é uma excelente plataforma para encontrar e trabalhar com datasets no formato **CSV**.

**Kaggle** é uma plataforma online de ciência de dados e machine learning, conhecida principalmente por hospedar competições, conjuntos de dados e notebooks interativos para análise e desenvolvimento de modelos de inteligência artificial.

### O que podes fazer no Kaggle?

1. **Participar em Competições** – Resolver desafios de machine learning com prémios em dinheiro e reconhecimento da comunidade.
2. **Explorar Conjuntos de Dados** – Aceder a milhares de datasets gratuitos para análise e modelagem.
3. **Executar Notebooks na Nuvem** – Usar Python e R diretamente no navegador sem precisar configurar um ambiente local.
4. **Aprender com Cursos Gratuitos** – A plataforma oferece cursos práticos de ciência de dados, estatística, Python, deep learning, entre outros.
5. **Colaborar com a Comunidade** – Partilhar código, insights e projetos com especialistas de todo o mundo.

## Parte III – SQLite em Python

### TEORIA

Uma **base de dados** é um conjunto organizado de informações estruturadas, armazenadas eletronicamente, que permite a gestão eficiente de grandes quantidades de dados. Estas podem ser utilizadas em diversas aplicações, como sistemas de gestão empresarial, sites, aplicações móveis, entre outros.

Um **Sistema de Gestão de Bases de Dados (SGBD)** é um software que permite criar, gerir e manipular bases de dados. Os principais objetivos de um SGBD incluem:

- ✓ **Armazenamento e recuperação eficiente de dados**
- ✓ **Garantia de integridade e consistência dos dados**
- ✓ **Controle de acessos e permissões**
- ✓ **Execução de operações complexas, como consultas e atualizações**

### Exemplos de Sistema de Gestão de Bases de Dados

Os SGBDs podem ser classificados em diferentes categorias, dependendo do modelo de dados utilizado:

- ✦ **Bases de Dados Relacionais (RDBMS)** → MySQL, PostgreSQL, Oracle, SQL Server, SQLite
- ✦ **Bases de Dados NoSQL** → MongoDB, Redis, Cassandra
- ✦ **Bases de Dados em Memória** → Memcached, Redis



As **Bases de Dados Relacionais (RDBMS - Relational Database Management Systems)** organizam os dados em **tabelas**, onde cada linha representa um registo e cada coluna representa um atributo. Estas bases de dados utilizam a linguagem **SQL (Structured Query Language)** para manipulação e gestão dos dados.

## 1. Principais conceitos de um SGBD Relacional

- **Tabelas** → Conjunto de dados organizados em colunas e linhas
- **Registos (Rows)** → Cada linha representa um conjunto de informações relacionadas
- **Campos (Columns)** → Cada coluna representa um atributo da informação
- **Chave Primária (Primary Key)** → Identificador único de cada registo numa tabela
- **Chave Estrangeira (Foreign Key)** → Ligação entre tabelas para manter a integridade relacional

O **SQLite** é um **SGBD relacional leve e embutido** que permite armazenar dados num ficheiro local sem necessidade de um servidor. Segue os princípios de bases de dados relacionais, mas de forma simplificada, tornando-se ideal para pequenas aplicações.

## 2. Principais características do SQLite

- ✓ **Baseado em Ficheiros** → Os dados são armazenados num único ficheiro `.sqlite` ou `.db`.
- ✓ **Não Requer Servidor** → Funciona diretamente com a aplicação, sem precisar de um sistema cliente-servidor.
- ✓ **Leve e Rápido** → Tem um desempenho excelente para pequenas e médias bases de dados.
- ✓ **Compatível com SQL** → Utiliza a linguagem SQL para manipulação de dados.
- ✓ **Portátil** → O ficheiro da base de dados pode ser facilmente transportado entre diferentes dispositivos.

### 3. Onde se utiliza SQLite?

- ◆ Aplicações móveis (Android, iOS)
- ◆ Aplicações web leves
- ◆ Software desktop
- ◆ Pequenos projetos locais
- ◆ Testes e prototipagem de bases de dados

### 4. Comparação entre SQLite e Outros SGBDs

Característica	SQLite	MySQL	PostgreSQL
Instalação	Nenhuma	Requer Servidor	Requer Servidor
Velocidade	Alta (para pequenas bases)	Média-Alta	Alta
Capacidade	Pequena a Média	Média a Grande	Grande
Uso	Local, apps móveis	Web, apps empresariais	Aplicações complexas
Transações	Sim	Sim	Sim
Suporte a JSON	Sim	Sim	Sim

### 5. Utilizar o SQLite em Python


O SQLite já vem incluído por padrão no Python, sendo possível usá-lo com a biblioteca `sqlite3`.

#### 1. Criar uma Base de Dados SQLite3 em Python

Antes de interagir com uma base de dados, precisas de a **criar e conectar**.

### **Sintaxe SQL**

sql



 Copy  Edit

```
CREATE DATABASE nome_da_base_de_dados;
```

★ **Nota:** No SQLite3, a base de dados é criada automaticamente ao conectar-se a um ficheiro que ainda não existe.

### **Exemplo em Python**

python

 Copy  Edit

```
import sqlite3

# Criar (ou conectar) uma base de dados chamada "universidade.db"
conexao = sqlite3.connect("universidade.db")

# Criar um objeto cursor para interagir com a base de dados
cursor = conexao.cursor()

print("Base de dados criada e conectada com sucesso!")



# Fechar a conexão
conexao.close()
```

## 2. Criar uma Tabela em SQLite3

As tabelas são onde os dados são armazenados dentro da base de dados.

### **Sintaxe SQL**

sql

 Copy  Edit

```
CREATE TABLE nome_da_tabela (
    coluna1 TIPO_DADO RESTRIÇÕES,
    coluna2 TIPO_DADO RESTRIÇÕES,
    ...
);
```

## Exemplo em Python

```
python                                                                    Copy Edit

import sqlite3

# Conectar à base de dados
conexao = sqlite3.connect("universidade.db")
cursor = conexao.cursor()

# Criar uma tabela "Alunos"
cursor.execute("""
CREATE TABLE IF NOT EXISTS Alunos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL,
    idade INTEGER,
    curso TEXT
)
""")

print("Tabela 'Alunos' criada com sucesso!")

# Fechar a conexão
conexao.commit()
conexao.close()
```

### 3. Inserir Dados na Tabela

Para adicionar novos registos à tabela, usamos o comando `INSERT INTO`.



## Sintaxe SQL

```
sql                                                                    Copy Edit

INSERT INTO nome_da_tabela (coluna1, coluna2, ...) VALUES (valor1, valor2, ...);
```

## Exemplo em Python

python

 Copy  Edit

```
import sqlite3

# Conectar à base de dados
conexao = sqlite3.connect("universidade.db")
cursor = conexao.cursor()

# Inserir alunos na tabela
cursor.execute("INSERT INTO Alunos (nome, idade, curso) VALUES (?, ?, ?)", ("João", 21, "Engenharia"))
cursor.execute("INSERT INTO Alunos (nome, idade, curso) VALUES (?, ?, ?)", ("Maria", 22, "Matemática"))
cursor.execute("INSERT INTO Alunos (nome, idade, curso) VALUES (?, ?, ?)", ("Carlos", 23, "Física"))

# Guardar e fechar a conexão
conexao.commit()
conexao.close()



print("Dados inseridos com sucesso!")
```

## 4. Ler Dados da Base de Dados

Podemos recuperar os dados de uma tabela utilizando o comando `SELECT`.

### Sintaxe SQL



sql

 Copy  Edit

```
SELECT * FROM nome_da_tabela;
```

## Exemplo em Python

python

 Copy  Edit

```
import sqlite3

# Conectar à base de dados
conexao = sqlite3.connect("universidade.db")
cursor = conexao.cursor()



# Selecionar todos os alunos
cursor.execute("SELECT * FROM Alunos")
alunos = cursor.fetchall() # Retorna todos os registos

# Exibir os alunos
for aluno in alunos:
    print(aluno)

# Fechar a conexão
conexao.close()
```

### ♦ Saída esperada:

bash

 Copy  Edit

```
(1, 'João', 21, 'Engenharia Informática')
(2, 'Maria', 22, 'Matemática Aplicada')
(3, 'Carlos', 23, 'Física')
```


## 5. Atualizar e Apagar Dados

Podemos modificar ou remover dados usando os comandos `UPDATE` e `DELETE`.

### Atualizar um registo

#### Sintaxe SQL

sql

 Copy  Edit

```
UPDATE nome_da_tabela SET coluna1 = novo_valor WHERE condição;
```

### 🌟 Exemplo em Python

```
python                                                                    Copy Edit

import sqlite3

conexao = sqlite3.connect("universidade.db")
cursor = conexao.cursor()

# Atualizar a idade do aluno "João"
cursor.execute("UPDATE Alunos SET idade = ? WHERE nome = ?", (24, "João"))

conexao.commit()
conexao.close()

print("Dados atualizados com sucesso!")
```

### 📌 Apagar um registo

#### 🌟 Sintaxe SQL

```
sql                                                                    Copy Edit

DELETE FROM nome_da_tabela WHERE condição;
```

### 🌟 Exemplo em Python

```
python                                                                    Copy Edit

import sqlite3

conexao = sqlite3.connect("universidade.db")
cursor = conexao.cursor()

# Apagar o aluno "Carlos"
cursor.execute("DELETE FROM Alunos WHERE nome = ?", ("Carlos",))

conexao.commit()
conexao.close()

print("Aluno apagado com sucesso!")
```

## 6. DOCUMENTAÇÃO

Para além dos comandos anteriores há muito mais que se pode fazer com a linguagem SQL e outros comandos. Se quiseres aprofundar consulta a documentação em <https://docs.python.org/3/library/sqlite3.html> e ainda o canal correspondente no Servidor do Discord.

## PRÁTICA

Resolve a seguinte atividade orientada. A ideia é replicar o código e efetuar a respetiva análise.

### ATIVIDADE: SQLite com Python

#### Objetivos

- Criar e gerir uma base de dados utilizando SQLite com Python.
- Executar operações básicas: inserção, consulta, atualização e eliminação de dados.
- Aplicar comandos SQL dentro de um script Python.



## Passo 1: Criar um ficheiro Python

Cria um ficheiro chamado `banco_dados.py` e adiciona o seguinte código base:

```
import sqlite3

# Criar conexão com o banco de dados (ou criar se não existir)
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()

# Criar tabela de funcionários
cursor.execute('''
    CREATE TABLE IF NOT EXISTS funcionarios (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nome TEXT NOT NULL,
        cargo TEXT NOT NULL,
        salario REAL NOT NULL
    )
''')

# Guardar as mudanças e fechar a conexão
conn.commit()
conn.close()
```

## Exercício 1:

1. Executa o ficheiro `banco_dados.py`.
2. Verifica se foi criado um ficheiro chamado `empresa.db` (esta é a base de dados SQLite).

## 2. Inserir Dados

Agora vamos inserir alguns funcionários na base de dados.

```
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()

# Inserir funcionários na tabela
cursor.execute("INSERT INTO funcionarios (nome, cargo, salario) VALUES ('Ana Silva', 'Gestora', 3500)
cursor.execute("INSERT INTO funcionarios (nome, cargo, salario) VALUES ('Pedro Santos', 'Programador')
cursor.execute("INSERT INTO funcionarios (nome, cargo, salario) VALUES ('Mariana Costa', 'Designer', 2500)

# Guardar mudanças e fechar conexão
conn.commit()
conn.close()
```

### Exercício 2:

1. Acrescenta mais dois funcionários à base de dados, alterando o código acima.
2. Executa o script e verifica se os dados foram adicionados.

## 3. Consultar Dados

Vamos agora recuperar os dados armazenados na tabela.

```
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()

# Consultar todos os funcionários
cursor.execute("SELECT * FROM funcionarios")
funcionarios = cursor.fetchall()

# Exibir os resultados
for funcionario in funcionarios:
    print(funcionario)

conn.close()
```

### Exercício 3:

1. Explica o que faz cada linha do código acima.
2. Executa o código e verifica os resultados.

## 4. Atualizar Dados

Vamos atualizar o salário de um funcionário específico.

```
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()

# Atualizar o salário de Pedro Santos
cursor.execute("UPDATE funcionarios SET salario = 3000.00 WHERE nome = 'Pedro Santos'")

conn.commit()
conn.close()
```

### Exercício 4:

1. Executa o código e verifica se o salário foi atualizado.
2. Modifica o código para aumentar o salário de todos os funcionários em 5%.

## 5. Eliminar Dados

Agora vamos remover um funcionário da base de dados.

```
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()

# Eliminar um funcionário
cursor.execute("DELETE FROM funcionarios WHERE nome = 'Mariana Costa'")

conn.commit()
conn.close()
```

### Exercício 5:

1. Executa o código e verifica se o funcionário foi eliminado.
2. Modifica o código para eliminar todos os funcionários com um salário inferior a 3000.00.

Cria um menu interativo para gerir a base de dados, onde o utilizador pode escolher entre: **1 Adicionar um novo funcionário** **2 Listar todos os funcionários** **3 Atualizar o salário de um funcionário** **4 Eliminar um funcionário** **5 Sair**

Tenta implementar o seguinte código:

```
def menu():
    while True:
        print("\nMENU DE GESTÃO DE FUNCIONÁRIOS")
        print("1. Adicionar funcionário")
        print("2. Listar funcionários")
        print("3. Atualizar salário")
        print("4. Eliminar funcionário")
        print("5. Sair")
        opcao = input("Escolha uma opção: ")

        if opcao == '1':
            nome = input("Nome: ")
            cargo = input("Cargo: ")
            salario = float(input("Salário: "))
            cursor.execute("INSERT INTO funcionarios (nome, cargo, salario) VALUES (?, ?, ?)", (nome

elif opcao == '2':
    cursor.execute("SELECT * FROM funcionarios")
    for funcionario in cursor.fetchall():
        print(funcionario)

elif opcao == '3':
    nome = input("Nome do funcionário: ")
    novo_salario = float(input("Novo salário: "))
    cursor.execute("UPDATE funcionarios SET salario = ? WHERE nome = ?", (novo_salario, nome

elif opcao == '4':
    nome = input("Nome do funcionário a eliminar: ")
    cursor.execute("DELETE FROM funcionarios WHERE nome = ?", (nome,))

elif opcao == '5':
    conn.commit()
    conn.close()
    break
else:
    print("Opção inválida! Tente novamente.")

# Criar conexão
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()
menu()
```