# Contents



# 1 Ficha técnica

**identificador**: workshop2021;

**título**: Overview sob o tema da Análise de Dados;

**data início**: 2021-12-15;


**autor**: José Carlos Ramalho, D1513;



**resumo**: Nesta workshop cobrem-se, sem aprofundar muito, os seguintes tópicos:

- Tipos de dados primitivos;
- Definição de funções;
- Tipos de dados estruturados: listas, tuplos e dicionários;
- Ficheiros: abertura, leitura, escrita e fecho;
- Leitura e limpeza de datasets: formatos CSV, JSON e XML;
- Cálculo de alguns indicadores e de algumas distribuições;
- Visualização gráfica de distribuições.


## 1.1 Tipos de dados primitivos

Inteiros, reais, complexos, booleanos e strings (???)

```
# Complexos
a = 7+4j
b = 3-2j
print(a+b)
print(a*b)
```

```
# Booleanos
a = 2*3 > 4
print(a)
b = (2*3 > 4) and (not a)
print(b)
```

```
# Reais
media = (13+18+12)/3
print(media)
```

```
# Strings
frase = "A alma humana é vítima tão inevitável da dor que sofre a dor da surpresa dolorosa"
# Acesso a um caráter
print(frase[7])
# Acesso a uma fatia
print(frase[2:6])
# A contar do fim: índices negativos
print(frase[-3:])
```

```
frase = "A alma humana é vítima tão inevitável da dor que sofre a dor da surpresa dolorosa"
# Comprimento
print("A frase tem " + str(len(frase)) + " carateres.")
print("Vamos iterar sobre a string:")
# E se quiser imprimir também a posição do caráter???
for c in frase:
    print(c)
```

```
# Partir uma string em substrings usando um ou mais caráteres como separador
frase = "A alma humana é vítima tão inevitável da dor que sofre a dor da surpresa dolorosa"
palavras = frase.split(" ")
print(palavras)
# O resultado é uma lista, vamos ver à frente...
```

## 1.2 Definição de funções

Sintaxe:

```
def nome(arg1, arg2, ...):
    instruções
    return resultado
```

```
def soma(a,b):
    return a+b

print(soma(7,12))
print(soma(soma(2,3),4))
```

### 1.2.1 Pequeno exercício

Vamos especificar uma função para calcular o resultado de elevar uma base a um determinado expoente.

```
def pot(b, e):
    res = 1
    # ...
    return res
```

## 1.3 Tipos de dados estruturados

### 1.3.1 Listas

Uma lista é uma sequência finita e ordenada de elementos. Um elemento pode ser qualquer coisa.

#### 1.3.1.1 Construtor:

```
# lista vazia
l = []
# lista homogénea
l2 = [1,2,3,4,5]
# lista heterogénea
l3 = [11, "onze", 12, "doze"]
# listas em compreensão
```

```
l4 = [x for x in (l+l2+l3)]
print(l4)
```

#### 1.3.1.2 Exercícios

```
# Lista com os inteiros no intervalo [1,100]

# Lista com os inteiros divisíveis por 7 no intervalo [1,1000]

frase = "A alma humana é vítima tão inevitável da dor que sofre a dor da surpresa dolorosa"
lista = [x for x in frase]
lista.sort()
print(lista)
```

#### 1.3.1.3 Comprimento

```
vogais = ['a', 'e', 'i', 'o', 'u']
print(len(vogais))
```

```
5
```

#### 1.3.1.4 *in* e *not in*

```
texto = "Hoje está um dia bonito!"
vogaisPresentes = []
for v in vogais:
    if v in texto:
        vogaisPresentes.append(True)
    else:
        vogaisPresentes.append(False)
print(vogaisPresentes)
```

#### 1.3.1.5 Acrescentar elementos numa lista

```
pares100 = []
i = 0
while i <= 100:
    pares100.append(i)
    i = i+2
print(pares100)

# Fazer o mesmo em compreensão
```

#### 1.3.1.6 Apagar o conteúdo duma lista

```
pares100.clear()
print(pares100)
```

#### 1.3.1.7 Copiar uma lista

```
cores = ["vermelho", "verde", "azul"]
cores2 = cores.copy() + ["amarelo"]
print(cores2)
print(cores2.index("verde"))
```

```
coresIndex = []
for i in range(len(cores2)):
    coresIndex.append((i, cores2[i]))
print(coresIndex)
```

#### 1.3.1.8 Inserir um elemento numa determinada posição

```
cores2.insert(1, "roxo")
print(cores2)
```

#### 1.3.1.9 Remover a primeira ocorrência de um elemento

```
cores2 = cores2 + ["verde", "verde"]
print(cores2)
cores2.remove("verde")
print(cores2)
```

#### 1.3.1.10 Ordem e ordenação

```
lista = [x for x in range(1,21)]
print(lista)
lista.reverse()
print(lista)
lista.sort()
print(lista)
```

### 1.3.2 Exercício: vamos criar uma lista...

```
def criarListaInt():
    lista = []
    n = int(input("Introduza o número de elementos da lista: "))
    i = 1
    while len(lista) < n:
        elem = int(input("Introduza o elemento " + str(i) + ": "))
        lista.append(elem)
        i = i+1
    return lista

l = criarListaInt()
print(l)
```

#### 1.3.2.1 Vamos automatizar a criação da lista

```
from random import randrange
def criarListaInt2(nelems):
    lista = []
    i = 1
    while len(lista) < nelems:
        elem = randrange(0, 101)
        lista.append(elem)
        i = i+1
    return lista

l2 = criarListaInt2(40)
print(l2)
l2.sort()
print(l2)
```

```
[100, 68, 94, 72, 86, 1, 29, 22, 60, 94, 26, 19, 77, 37, 65, 30, 19, 3, 99, 37, 69, 17, 60, 3, 64, 28,
[1, 3, 3, 17, 19, 19, 20, 21, 22, 25, 26, 28, 29, 30, 30, 37, 37, 45, 52, 57, 60, 60, 63, 64, 65, 67, 6
```

#### 1.3.2.2 Há duplicados, como podemos evitar que sejam gerados?

```
# Resolver na sessão
```

#### 1.3.2.3 Exercício: Vamos definir uma função para calcular o maior da lista

## 1.4 Tuplos e companhia

#### 1.4.0.1 Construtores: (), tuple()

```
aluno1 = ("Ana Maria", "A87564", "ENGMAT", 1,1,0,0,1,1,0,0)
coordenadas = tuple([39, 9])
```

#### 1.4.0.2 Acesso

```
curso1 = aluno1[2]
print(curso1)
print(coordenadas[1])
```

#### 1.4.0.3 Acesso com desmembramento (unfold/unpack)

```
nome, id, curso, *tpc = aluno1
print(id + ": " + nome)
print(tpc)
```

#### 1.4.0.4 Testar se um valor está no tuplo

```
if "ENGMAT" in aluno1:
    print("Temos mais um aluno!")
```

#### 1.4.0.5 Remover um campo

```
a1Lista = list(aluno1)
a1Lista.pop(2)
aluno1 = tuple(a1Lista)
print(aluno1)
```

## 1.5 Listas em compreensão

```
letras = [l for l in "A Maria tem um namorado"]
print(letras)
```

```
['A', ' ', 'M', 'a', 'r', 'i', 'a', ' ', 't', 'e', 'm', ' ', 'u', 'm', ' ', 'n', 'a', 'm', 'o', 'r', 'a
```

#### 1.5.0.1 Sintaxe: [expressão for elem in lista]

```
quadrados = [n*n for n in range(20)]
print(quadrados)
```

```
parImpar = [("Par", y) if y%2 == 0 else ("Ímpar", y) for y in range(1,101)]
print(parImpar)
```

#### 1.5.0.2 Exercício: Especifica uma lista em compreensão com os inteiros entre 0 e 1000 divisíveis por 2 e por 7

#### 1.5.0.3 Exercício: Resultado após uma jornada de futebol

```
jornada21 = ('Jornada XXI', [(('Benfica',1),('Moreirense',3)),
                             (('Sporting',2),('Braga',3)),
                             (('Porto',3),('Rio Ave', 3))])
print(jornada21)
```

```
# Devolve uma lista de pares: [(Equipe, Pontos)]
def resultado(jornada):
    resultado = []
    for jogo in jornada21[1]:
        visitado, visitante = jogo
        if visitado[1] > visitante[1]:
            resultado.append((visitado[0],3))
            resultado.append((visitante[0],0))
        elif visitado[1] < visitante[1]:
            resultado.append((visitado[0],0))
            resultado.append((visitante[0],3))
        else:
            resultado.append((visitado[0],1))
            resultado.append((visitante[0],1))
    return resultado
```

```
def verResultado(rlista):
    for tuplo in rlista:
        equipe, pontos = tuplo
        print(equipe, " - ", pontos)
```

## 1.6 Dicionários

#### 1.6.0.1 Construtores: {}, dict()

```
jogo1 = {'visitado': {'equipe': 'Benfica', 'golos': 1},
        'visitante': {'equipe': 'Moreirense', 'golos': 3}}
print(jogo1)
print(jogo1['visitado']['golos'])
print(jogo1.get('visitado').get('equipe'))
```

#### 1.6.0.2 Acesso

```
compras = {'maçã': 5, 'laranja':6, 'banana': 7}
print(compras['laranja'])
print(compras.get('banana'))
```

#### 1.6.0.3 Extrair chaves

```
distrib = {"LCC": 23, "ENGBIOM": 35, "LEI": 32, "ENGFIS": 17}
chaves = distrib.keys()
print(type(chaves))
print(chaves)
```

#### 1.6.0.4 Extrair valores

```
distrib = {"LCC": 23, "ENGBIOM": 35, "LEI": 32, "ENGFIS": 17}
valores = distrib.values()
print(type(valores))
print(valores)
```

#### 1.6.0.5 Concatenar dicionários

```
legumes = {"cenouras": 6, "batatas":20, "cebolas": 12}
frutas = {"tangerina": 30, "pêras":8, "bananas": 6, "romãs": 2}
charcutaria = {"fiambre": 10, "queijo": 16, "chouriço": 1}
listaCompras = {}
for par in (legumes, frutas, charcutaria): listaCompras.update(par)
print(listaCompras)
```

```
legumes = {"cenouras": 6, "batatas":20, "cebolas": 12}
frutas = {"tangerina": 30, "pêras":8, "bananas": 6, "romãs": 2}
charcutaria = {"fiambre": 10, "queijo": 16, "chouriço": 1}
listaCompras = legumes.copy()
listaCompras.update(frutas)
listaCompras.update(charcutaria)
print(listaCompras)
```

```
legumes = {"cenouras": 6, "batatas":20, "cebolas": 12}
frutas = {"tangerina": 30, "pêras":8, "bananas": 6, "romãs": 2}
charcutaria = {"fiambre": 10, "queijo": 16, "chouriço": 1}
listaCompras = {}
for chave in legumes.keys():
    listaCompras[chave] = legumes[chave]
for chave in frutas.keys():
    listaCompras[chave] = frutas[chave]
for chave in charcutaria.keys():
    listaCompras[chave] = charcutaria[chave]
print(listaCompras)
```

#### 1.6.0.6 Ordenar dicionários: por valor

```
def ordenaValores(v):
    return v[1]

# print(listaCompras.items())
# valores = sorted(listaCompras.items(), key = lambda x: x[1])
valores = list(listaCompras.items())
valores.sort(key = ordenaValores)
novoDict = dict(valores)
print(novoDict)
```

#### 1.6.0.7 Ordenar dicionários: por chave

```
def ordenaChaves(par):
    return par[0]

chaves = sorted(listaCompras.items(), key = ordenaChaves)
# Fazer com o sort
print(chaves)
```

#### 1.6.0.8 Verificar se uma chave está no dicionário

```
def pertenceChave(c, d):
    return True if c in d else False

print(pertenceChave("bananas", listaCompras))
print(pertenceChave("beterrabas", listaCompras))
```

**1.6.0.9 Exercício:** Escreve um programa em Python que leia um número inteiro positivo e crie um dicionário com chaves de 1 até esse número, em que o valor associado a cada chave é o quadrado dessa chave.

```
# Desenvolver na workshop
```

#### 1.6.0.10 O problema da inversão estrutural

```
escola = [
    {
        'nome': "Ana",
        'instrumentos': ['trompete','clarinete', 'flauta']
    },
    {
        'nome': "Carlos",
        'instrumentos': ['saxofone', 'flauta', 'piano']
    },
    {
        'nome': "Paulina",
        'instrumentos': ['saxofone', 'violino', 'piano']
    }
]

def calcListaInst(esc):
    instrumentos = []
    for a in esc:
        for i in a['instrumentos']:
            if i not in instrumentos:
                instrumentos.append(i)
    instrumentos.sort()
    return instrumentos

def calcListaMusicos(esc, i):
    musicos = []
    for a in esc:
```

```
        if i in a['instrumentos']:
            musicos.append(a['nome'])
    return musicos

def indInstrumentos(esc):
    indiceInstrumentos = []
    listaInst = calcListaInst(esc)
    for i in listaInst:
        indiceInstrumentos.append({
            'instrumento': i,
            'musicos': calcListaMusicos(esc, i)
        })
    return indiceInstrumentos

print(indInstrumentos(escola))
```

## 1.7 Persistência da informação / Armazenamento em memória secundária: ficheiros

### 1.7.1 Abertura: open(nome, modo)

Modos de abertura de um ficheiro:

- "r" - Read - Valor por omissão. Abre o ficheiro para leitura, devolve error se o ficheiro não existir;
- "a" - Append - Abre o ficheiro para acrescentar, cria o ficheiro se este não existir;
- "w" - Write - Abre o ficheiro para escrever, cria o ficheiro se este não existir, apaga o conteúdo se este existir;
- "x" - Create - Cria o ficheiro, devolve error se o ficheiro já existir.

Parâmetro adicional ao modo:

- "t" - Text - Valor por omissão. Ficheiro textual;
- "b" - Binary - Ficheiro binário, por exemplo, imagens.

```
f = open("bdArtigos.json", "rt")
f2 = open("bdArtigos.json", "r")
f3 = open("bdArtigos.json")
```

### 1.7.2 Leitura do conteúdo

```
f4 = open("demo.txt")
print(f4.read())
```

#### 1.7.2.1 Uma linha de cada vez, iterando pelas linhas...

```
f4 = open("demo.txt")
print(f4.readline())
print(f4.readline())
```

```
f4 = open("demo.txt")
i=1
for linha in f4:
    print(str(i) + " :: " + linha)
    i = i+1
```

### 1.7.3 Fechar um ficheiro: devemos fazê-lo no fim...

```
f4 = open("demo.txt")
i=1
for linha in f4:
    print(str(i) + " :: " + linha)
    i = i+1
f4.close()
```

### 1.7.4 Escrever num ficheiro

```
f5 = open("demo.txt", "a")
f5.write("Agora o ficheiro tem mais conteúdo!")
f5.close()

f5 = open("demo.txt", "r")
print(f5.read())
```

### 1.7.5 Exercício: uma BD bibliográfica - Como guardar em ficheiro?

```
arts = [{'chaveCit': 'RHA2008', 'titulo': 'Álgebra Documental',
        'autores': [('jcr', 'José Carlos Ramalho'), ('prh', 'Pedro Rangel Henriques'), ('jj', 'José Joã
        'ano': 2008, 'local': 'Twente - Holanda'},
    {'chaveCit': 'RHA1998', 'titulo': 'Utilização do email pedagogicamente',
        'autores': [('jcr', 'José Carlos Ramalho'), ('prh', 'Pedro Rangel Henriques'), ('jj',
        'José João Almeida')], 'ano': 1999, 'local': 'Universidade do Minho'}]
```

### 1.7.6 Hipótese 1: no "meu" ficheiro de texto

**Que forma irão ter os registos?**

`chaveCit::titulo::autor1#autor2#...#autorn::ano::local`

Aplicado ao nosso exemplo:

'RHA2008::Álgebra Documental::jcr,José Carlos Ramalho#prh,Pedro Rangel Henriques#jj,José João Almeida::2008::Twente - Holanda

RHA1998::Utilização do email pedagogicamente::jcr,José Carlos Ramalho#prh,Pedro Rangel Henriques#jj,José João Almeida::1999::Universidade do Minho'

```
# Guardar a BD no ficheiro
def guardarBD(bd, fnome):
    f = open(fnome, 'w', encoding='utf-8')
    sep = "::"
    myReg = ""
    for art in arts:
        myReg = art['chaveCit'] + sep + art['titulo'] + sep
        i = 1
        for (id, nome) in art['autores']:
            myReg = myReg + id + "," + nome
            if i < len(art['autores']):
                myReg = myReg + "#"
            else:
                myReg = myReg + "::"
            i = i+1
        myReg = myReg + str(art['ano']) + sep + art['local']
        myReg = myReg + "\n"
        f.write(myReg)

guardarBD(arts, "bdArtigos.txt")
print("Foram guardados " + str(len(arts)) + " registos.")
```

```
# Carregar a BD dum ficheiro
def extraiAutores(linha):
    res = []
    autores = linha.split("#")
    for a in autores:
        acampos = a.split(",")
        res.append((acampos[0],acampos[1]))
    return res

def carregarBD(fnome):
    bd = []
```

```
    f = open(fnome)
    for linha in f:
        myReg = {}
        campos = linha.split("::")
        myReg['chaveCit'] = campos[0]
        myReg['titulo'] = campos[1]
        myReg['autores'] = extraiAutores(campos[2])
        myReg['ano'] = campos[3]
        myReg['local'] = campos[4]
        bd.append(myReg)
    return bd

teste = carregarBD("bdArtigos.txt")
print(teste)

# Mais umas reutilizações do código já escrito
# Inserir mais alguns registos e guardar
# ...
```

### 1.7.7 Hipótese 2: em JSON

(fazer uma visita guiada a json.org)

```
# Guardar a BD em JSON
import json

def guardarBD(bd, fnome):
    f = open(fnome, 'w', encoding='utf-8')
    json.dump(bd, f, ensure_ascii=False, indent=4)

guardarBD(arts, "bdArtigos.json")

# Carregar a BD dum ficheiro em JSON
def carregarBD(fnome):
    f = open(fnome)
    bd = json.load(f)
    return bd

teste = []
teste = carregarBD("bdArtigos.json")
print(teste)
```

O campo autores não veio no formato pretendido. *Porquê?...*

Altera a função de carregamento de modo a colocar tudo no formato esperado.

## 1.8 Plot: desenho de gráficos e imagens

Para a representação gráfica iremos usar o módulo plot.

### 1.8.1 Instalação

```
pip install matplotlib
```

### 1.8.2 Plot1: uma espécie de "Hello world!"

```
import matplotlib.pyplot as plt

# Valores para o eixo do X
x = [1,2,3]
# Valores para o eixo do Y
y = [2,4,1]

# Marcar os pontos
```

```
plt.plot(x, y)

plt.xlabel('Abcissas')
plt.ylabel('Ordenadas')
plt.title('O meu primeiro gráfico!')

# Desenhar o gráfico
plt.show()
```

### 1.8.3 Plot2: duas funções no mesmo gráfico

```
import matplotlib.pyplot as plt
# função 1
x1 = [1,2,3]
y1 = [2,4,1]
plt.plot(x1, y1, label = "linha 1")

# função 2
x2 = [1,2,3]
y2 = [4,1,3]
plt.plot(x2, y2, label = "linha 2")

plt.xlabel('Abcissas')
plt.ylabel('Ordenadas')
plt.title('Gráfico com duas funções')
plt.legend()

plt.show()
```

### 1.8.4 Plot3: configurações

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6]
y = [2,4,1,5,2,6]

plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)

# Limites dos eixos
plt.ylim(1,8)
plt.xlim(1,8)

plt.xlabel('Abcissas')
plt.ylabel('Ordenadas')
plt.title('Um gráfico com estilo!')

plt.show()
```

### 1.8.5 Exercício: desenha a função `f = x^2 - 7`

```
# ...
```

## 1.9 Gráficos de barras

```
import matplotlib.pyplot as plt

# coordenada-x do lado esquerdo das barras
left = [1, 2, 3, 4, 5]

# altura das barras
height = [6, 4, 8, 12, 5]
```

```
# labels das barras
tick_label = ['cebolas', 'maçãs', 'pêras', 'batatas', 'cenouras']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.8, color = ['red', 'green'])

plt.xlabel('Frutas e legumes')
plt.ylabel('Quantidade')
plt.title('O meu stock de verdes')

plt.show()
```

#### 1.9.0.1 Exercício: faz o gráfico de barras para a lista de compras

```
legumes = {"cenouras": 6, "batatas":20, "cebolas": 12}
frutas = {"tangerina": 30, "pêras":8, "bananas": 6, "romãs": 2}
listaCompras = legumes.copy()
listaCompras.update(frutas)
print(listaCompras)
```

## 1.10 Distribuições: uma introdução...

Neste problema, vamos calcular indicadores estatísticos sobre o dataset dos Exames Médicos Desportivos:

- Distribuição por sexo;
- Distribuição por modalidade;
- Distribuição por escalão etário: menos de 21, entre 21 e 35, e mais de 35.