PRÁTICA LABORATORIAL 05

Objectivos:
Listas, tuplos, matrizes, dicionários, List Comprehension, estruturas de repetição, módulos, try,except, funções

Funções, Módulos e Tratamento de Exceções

Índice

1. Funções em Python

Explicação Teórica

Funções são blocos de código reutilizáveis que executam uma tarefa específica. Permitem escrever código modular, legível e organizado.

Sintaxe básica:

```
def nome_da_funcao(parametros):
    # bloco de código
    return resultado
```

Para que servem:

- Evitar repetição de código
- Organizar a lógica do programa
- Tornar o código mais limpo e legível
- Ajudar na manutenção e teste do código

Exemplos

```
def saudacao():
    print("Olá, bem-vindo!")
saudacao()

def soma(a, b):
    print(a + b)
soma(5, 3)

def multiplica(a, b):
    return a * b
resultado = multiplica(4, 5)
print(resultado)

def cumprimenta(nome="utilizador"):
    print(f"Olá, {nome}!")
cumprimenta()
cumprimenta("Ruben")

def eh_par(numero):
    return numero % 2 == 0
print(eh_par(4))  # True
```

Perguntas Frequentes

- Para que serve o return? → Devolve um valor da função para o ponto onde foi chamada.
- Posso chamar uma função dentro de outra? → Sim, é uma prática comum e útil.
- O que acontece se não usar return? → A função retorna None por padrão.
- Posso definir funções com nome igual a comandos do Python? → Não é recomendado, pois sobrescreve a função original.
- Posso passar listas ou dicionários como parâmetro? → Sim, qualquer tipo de objeto pode ser passado como argumento.

Curiosidade: Em Python, funções também são objetos! Podem ser atribuídas a variáveis, passadas como argumento e até retornadas por outras funções.

```
# 1. Atribuir uma função a uma variável
def cumprimentar(nome):
    return f"Olá, {nome}!"

f = cumprimentar  # A função é atribuída à variável f
print(f("Maria"))  # Olá, Maria!

# 2. Passar função como argumento
def executa(funcao, valor):
```

```
    return funcao(valor)

print(executa(cumprimentar, "João"))  # Olá, João!

# 3. Retornar função a partir de outra
def escolher_operacao(nome):
    def ola():
        return f"Olá, {nome}!"
    return ola

mensagem = escolher_operacao("Ana")
print(mensagem())  # Olá, Ana!
```

2. Módulos em Python

python3 -m pip list para verificar módulos existentes

Explicação Teórica

Módulos são ficheiros .py que contêm funções, variáveis e classes. Permitem dividir o código em múltiplos ficheiros para melhor organização e reutilização.

Importação básica:

```
import math
from math import sqrt
```

Para que servem:

- Organizar código em diferentes ficheiros
- Reutilizar código em diferentes projetos
- Aceder a bibliotecas já construídas

Exemplos

Import datetime – retorna dados sobre hora e data

```
dhora = datetime.datetime.now()
```

```
datahora_atual.year         # Ano
datahora_atual.month        # Mes
datahora_atual.day          # dia
datahora_atual.hour         # Hora
datahora_atual.minute       # Minuto
datahora_atual.second       # Segundo
datahora_atual.microsecond  # Microsegundo
```

```
import math
print(math.sqrt(16))  # 4.0

from math import pi
print(pi)  # 3.14159...
```

```
import random as rd
print(rd.randint(1, 10))

# meumodulo.py
def saudacao(nome):
    return f"Olá, {nome}!"
# script principal
import meumodulo
print(meumodulo.saudacao("Ana"))
```

O comando print(dir(math)) em Python serve para **listar todos os nomes (funções, constantes, classes, etc.) que estão disponíveis no módulo math**.

```
import                                                    math
print(dir(math))
```

Perguntas Frequentes
• O que é um módulo? → É um ficheiro Python com funções e variáveis reutilizáveis.
• Qual a diferença entre import e from ... import? → O primeiro importa tudo, o segundo importa elementos específicos.

```
from math import sqrt, pi

print(sqrt(16))  # Não precisa de math.
print(pi)       # Também não precisa de math.
```

• O que é 'as' no import? → Serve para dar um nome alternativo ao módulo.

```
import numpy as np
array = np.array([1, 2, 3])
print(array)

from math import factorial as fat
print(fat(5))  # Resultado: 120
```

• Posso importar módulos de outros projetos? → Sim, se estiverem no mesmo diretório ou no PYTHONPATH.
• Posso criar os meus próprios módulos? → Sim, basta criar um .py com funções ou variáveis.

Criar boasvidas.py

```
Def func(nome):
        Print("bemvindo {nome}
```

Import boasvindas.py
Boasvindas.func("Ruben")

Curiosidade: Python vem com mais de 200 módulos padrão, conhecidos como standard library.

3. Try / Except / else /Finally

Explicação Teórica

Esta estrutura serve para lidar com erros (exceções) de forma controlada, evitando que o programa termine abruptamente.

Estrutura básica:

try:
  # bloco que pode dar erro
except TipoDeErro:
  # o que fazer se houver erro
Else:
  # se não der erro
finally:
  # sempre executado (opcional)

```
try:
    x = int('Python')
except:
    print('Não é possivel converter.')
else:
    print('Convertido com sucesso.')
finally:
    print('Processo terminado.')
```

Exemplos

```
try:
  x = 10 / 0
except:
  print("Erro ocorreu.")

try:
  numero = int("abc")
except ValueError:
  print("Valor inválido.")

try:
  lista = [1, 2]
  print(lista[5])
except IndexError:
  print("Índice fora do intervalo.")
except Exception as e:
  print("Erro desconhecido:", e)

try:
  f = open("ficheiro.txt")
```

```
except FileNotFoundError:
    print("Ficheiro não encontrado.")
finally:
    print("Bloco finally executado.")

try:
    resultado = 10 / 2
except ZeroDivisionError:
    print("Divisão por zero.")
else:
    print("Resultado:", resultado)
```

fazer com que uma função resulte em erro personalizado

```
def verificar_nome(nome):
    if not nome.strip():  # Verifica se o nome está vazio ou só tem espaços
        raise ValueError(" O nome não pode estar vazio!")
    print(f" Nome válido: {nome}")

# Programa principal
try:
    verificar_nome("")  # Teste com string vazia
except ValueError as erro:
    print(f"Ocorreu um erro: {erro}")
```

finally

Perguntas Frequentes

• O que é uma exceção? → É um erro que ocorre durante a execução do programa.
• O que acontece se não usar except? → O programa para imediatamente se ocorrer um erro.
• Posso capturar todos os erros com except? → Sim, mas o ideal é capturar erros específicos.
• O bloco finally é sempre executado? → Sim, mesmo que haja erro ou return.
• Qual a diferença entre except e finally? → except lida com erros; finally é usado para limpeza e será sempre executado.

Curiosidade: O Python permite até criar exceções personalizadas com classes que herdam de Exception.

Exercícios

1. Crie um programa em Python que solicite ao utilizador a introdução de 10 valores inteiros, armazene-os numa lista, e apresente os seguintes resultados no ecrã:
   1.1. Contabilizar o número de valores que são negativos;
   1.2. Calcular a média dos valores positivos inseridos;
   1.3. Identificar os valores que são múltiplos de 3 e de 5 em simultâneo.

2. Crie um programa em Python que leia duas horas completas (hora de entrada e hora de saída de um funcionário), e calcule o tempo total em que esse funcionário esteve a trabalhar. Para isso:

2.1. Solicite ao utilizador que insira 3 valores inteiros (hora, minuto e segundo) para representar a hora de entrada e a hora de saída;
2.2. Crie uma função que receba os 3 valores (hora, minuto e segundo) como parâmetros, e converta tudo para segundos, retornando esse valor;
2.3. Subtraia os valores em segundos das duas horas inseridas, obtendo assim o intervalo em segundos;
2.4. Crie uma nova função que receba esse intervalo (em segundos) e converta-o para o formato HH:MM:SS, apresentando o resultado final no ecrã.

3. Crie um programa em Python que leia um conjunto de idades de uma turma de N alunos, guarde-as numa lista, e chame uma função que determine a idade do aluno mais velho.
3.1. A função deve receber a lista de idades como parâmetro;
3.2. A função deve retornar a idade mais elevada encontrada.

4. Crie um programa em Python que leia um conjunto de alturas de N pessoas, armazene os valores numa lista, e chame uma função que conte quantas pessoas têm altura entre 1.50m e 1.75m (inclusive).
4.1. A função deve receber a lista de alturas como parâmetro;
4.2. A função deve retornar o número de pessoas dentro do intervalo especificado.

5. Crie um programa em Python que leia um conjunto de NIFs de N pessoas, armazene-os numa lista, e chame uma função que verifique a posição da lista onde se encontra um NIF fornecido pelo utilizador.
5.1 A função deve receber a lista de NIFs e o NIF a pesquisar;
5.2. A função deve retornar a posição (índice) do NIF, ou lançar uma exceção caso não exista.
5.3. Utilize try/except para validar a introdução de números inteiro e também para capturar o erro.

6. Crie um programa em Python que simule N lançamentos de dois dados, armazene os resultados dos lançamentos numa lista de tuplos (por exemplo, (3, 5) para um lançamento), e gere uma estatística que mostre qual o valor total mais comum (soma dos dois dados).
6.1. Utilize um dicionário para contabilizar a frequência de cada soma;
6.2. Apresente a soma mais comum no final;
6.3. Implemente o programa com tratamento de exceções para validar o número de lançamentos.

7. Crie um programa em Python que leia uma lista de N valores inteiros, substitua todos os valores negativos por 0, e apresente:
7.1. A lista resultante;
7.2. O número total de valores alterados;
7.3. Inclua validação de dados usando try/except.

8. Crie um programa em Python que leia N valores inteiros sem permitir repetição, utilizando um módulo personalizado para realizar as validações e operações principais.

Etapas:

- Crie um ficheiro validador.py com:
- Uma função ler_valores_unicos(n) que:
    - Receba o número de valores a ler;
    - Utilize um set para garantir unicidade;
    - Converta os valores para list mantendo a ordem de inserção;
    - Utilize try/except para validar se o valor inserido é um inteiro.
- No ficheiro principal programa.py, importe a função do módulo e chame-a.

9. Crie um programa em Python que leia uma lista de N valores inteiros e determine o segundo maior valor. Organize o código em dois ficheiros:

Etapas:

- Crie um ficheiro operacoes.py com:
- Uma função ler_lista_inteiros(n) com try/except para validar entradas;
- Uma função segundo_maior(lista) que:
    - Verifique se existem pelo menos dois valores distintos;
    - Retorne o segundo maior valor ou mensagem de erro.
- No ficheiro principal programa.py, importe as funções do módulo e utilize-as.

Bom trabalho!