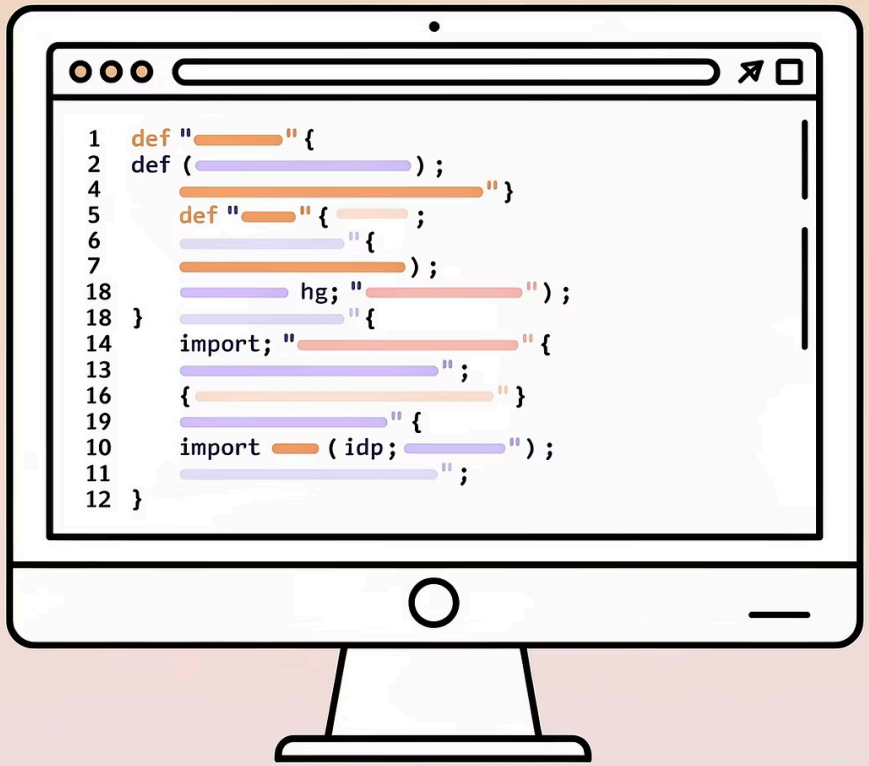


CEFET/RJ

Listas em Python

Listas em Python são estruturas de dados **ordenadas e mutáveis** que podem armazenar elementos de tipos diferentes no mesmo objeto. Elas são usadas para agrupar dados relacionados de forma organizada e dinâmica.

Uma lista pode conter números, strings, vetores NumPy, DataFrames, funções ou até outras listas.

Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

FUNDAMENTOS

O que são listas?

Uma lista em Python é uma **coleção ordenada de objetos**. Os elementos podem se repetir e são acessados por índice, começando em zero.

Cada elemento pode ter um tipo diferente, sem restrição de homogeneidade. Isso torna as listas extremamente versáteis e poderosas.

```
my_list = [1, "a", 3.5, True]
my_list
```

Saída esperada:

```
[1, 'a', 3.5, True]
```

CRIAÇÃO

Criando listas

Sintaxe básica

Listas são criadas usando colchetes [], separando os elementos por vírgulas

Tipos misturados

Podem conter vetores NumPy, strings e números ao mesmo tempo

Aninhamento

É possível aninhar listas dentro de listas para estruturas complexas

```
import numpy as np
weight = np.array([60, 72, 57, 90, 95, 72])
height = np.array([1.75, 1.80, 1.65, 1.90, 1.74, 1.91])
subject = ["A", "B", "C", "D", "E", "F"]
mybag = [weight, height, subject, 0, "a"]
len(mybag), type(mybag)
```

Saída esperada: (5, <class 'list'>)

ESTRUTURA

Conteúdo da lista

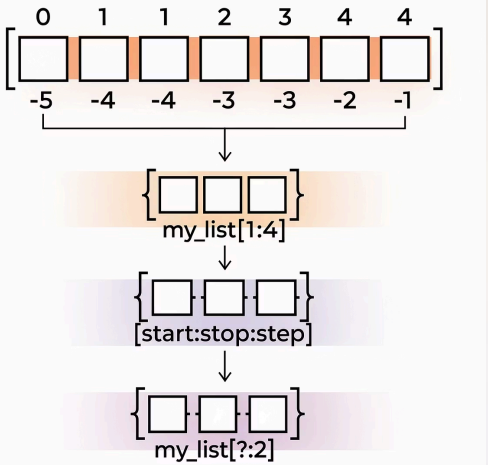
Uma lista pode conter **múltiplos objetos de tipos diferentes**. Cada posição mantém o tipo original do objeto armazenado.

Os elementos são acessados pelo índice, permitindo navegação precisa através da estrutura de dados.

```
for i, item in enumerate(mybag):
    print(i, type(item))
```

Saída esperada:

```
0 <class 'numpy.ndarray'>
1 <class 'numpy.ndarray'>
2 <class 'list'>
3 <class 'int'>
4 <class 'str'>
```



INDEXAÇÃO

Fatiamento de listas

01

Indexação zero-based

Listas são indexadas a partir de zero, o primeiro elemento está na posição 0

02

Padrão de fatiamento

O fatiamento usa o padrão início incluso e fim excluso

03

Exemplo prático

`[0:3]` retorna os elementos nas posições 0, 1 e 2

```
mybag[0:3]
```

Retorna: array de peso, altura e lista de sujeitos

```
mybag[:2]
```

Retorna: apenas arrays de peso e altura

DIFERENÇAS IMPORTANTES

Elemento vs sublista

Acesso por índice único

Quando usamos um **índice único**, obtemos o objeto armazenado diretamente na lista

```
mybag[0]  # Retorna: numpy.ndarray
```

Acesso por intervalo

Quando usamos um **intervalo**, obtemos uma sublista contendo os elementos selecionados

```
mybag[0:1]  # Retorna: list
```

Exemplo completo:

```
print(type(mybag[0:1]), mybag[0:1])
print(type(mybag[0]), mybag[0])
```

Saída:

```
<class 'list'> [array([60, 72, 57, 90, 95, 72])]
<class 'numpy.ndarray'> [60 72 57 90 95 72]
```

Essa diferença é fundamental para entender a estrutura dos dados e evitar erros comuns.

Python dataclass

ESTRUTURAS AVANÇADAS

Registros com dataclass

Quando um conjunto de dados tem **campos fixos e semânticos**, Python usa `dataclass`. Ela define um tipo com atributos nomeados, acessados por ponto.

Isso representa registros estruturados, não apenas coleções genéricas, trazendo mais clareza e organização ao código.

```
from dataclasses import dataclass
import numpy as np

@dataclass
class Bag:
   weight: np.ndarray
   height: np.ndarray
   subject: list
   valor: int
   nome: str

mybag = Bag(weight, height, subject, 0, "a")
mybag.weight
```

Saída: array([60, 72, 57, 90, 95, 72])

FLEXIBILIDADE

Campos opcionais

Adicionar atributos

Objetos dataclass podem ganhar novos atributos durante a execução

Marcar como ausente

Campos irrelevantes podem ser marcados como `None`

Manter consistência

Isso mantém a estrutura explícita e consistente

```
mybag.bmi = mybag.weight / (mybag.height ** 2)
mybag.valor = None
mybag.nome = None
mybag
```

Resultado: Objeto Bag com novo campo `bmi` calculado e campos `valor` e `nome` definidos como None

CUIDADO!

Indexação e mutabilidade

Mutabilidade por referência

Listas e objetos são **mutáveis por referência**. Isso significa que múltiplas variáveis podem apontar para o mesmo objeto na memória.

Cópia rasa

A cópia de uma lista usando `.copy()` é rasa, então objetos internos continuam compartilhados entre as listas.

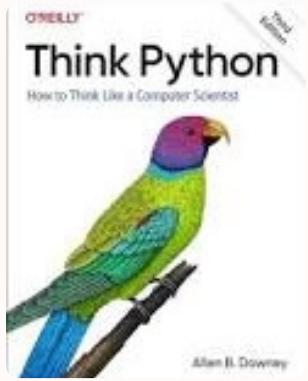
Alterações em objetos internos afetam todas as referências.

```
mybag_list = [weight, height, subject]
a = mybag_list
b = mybag_list.copy()
a[0][0] = 999
a[0][0], b[0][0]
```

Saída esperada: `(999, 999)`

Ambos os valores mudaram porque compartilham a mesma referência ao array NumPy!

Referências



Think Python

Downey, A. *Think Python: How to Think Like a Computer Scientist.* O'Reilly Media.

An essential introduction to programming fundamentals and computational thinking using Python.

Python Data Science Handbook

VanderPlas, J. *Python Data Science Handbook.* O'Reilly Media.

A comprehensive guide to essential tools for working with data in Python, including NumPy, Pandas, and visualization libraries.

Data Science from Scratch

Grus, J. *Data Science from Scratch.* O'Reilly Media.

Learn data science fundamentals by building algorithms and tools from the ground up using Python.