

# python™

## OS MISTÉRIOS DO BACKEND PYTHON:

### DESCOBRIENDO A MAGIA DOS DADOS EM HOGWARTS



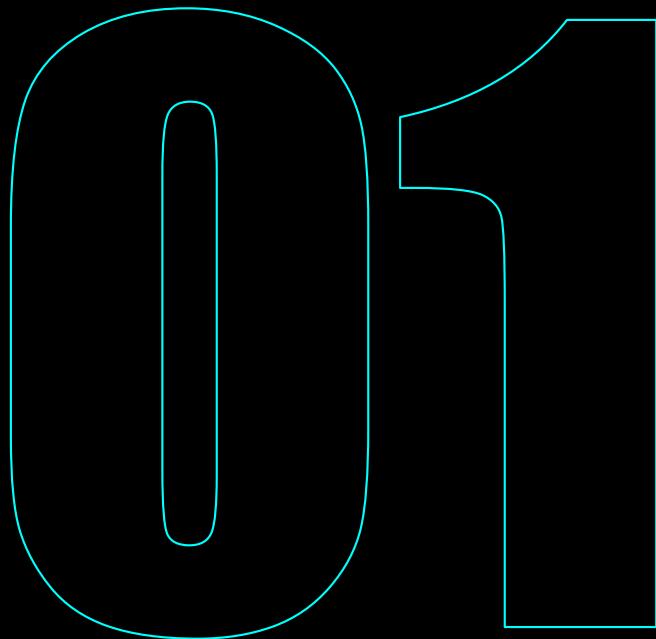
DAYANA FERREIRA

# ESTRUTURAS DE DADOS

## Explorando as Maravilhas das Coleções de Dados em Python

Se você já se aventurou pelo mundo da programação em Python, provavelmente já ouviu falar sobre coleções de dados. Elas são como caixinhas mágicas onde podemos armazenar nossas informações de forma organizada e acessá-las sempre que precisarmos. Vamos dar uma olhada mais de perto em algumas das coleções mais poderosas que Python oferece: listas, tuplas, conjuntos e dicionários.





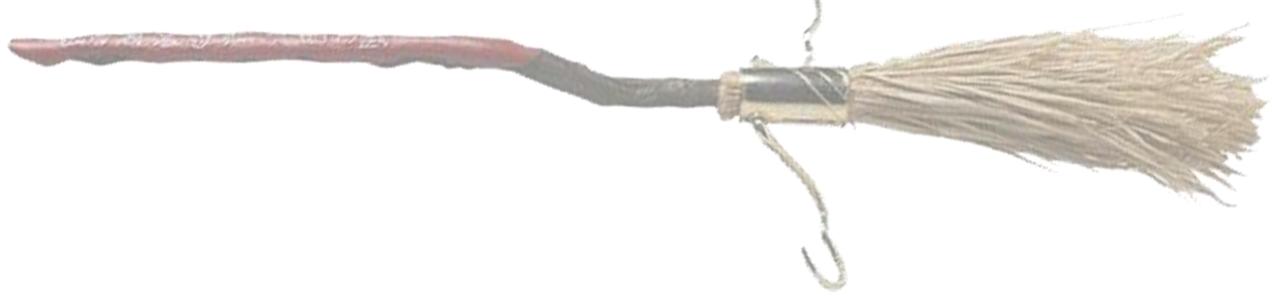
# LISTAS

---

SEU MELHOR AMIGO NA ORGANIZAÇÃO DE DADOS

Imagine-a como uma mochila mágica onde você pode guardar uma infinidade de itens - e, mais importante, pode modificá-la conforme desejar.

# CONCEITOS ESSENCIAIS



As listas são como uma sequência ordenada de itens, onde cada item é acessado por sua posição na lista, chamada de índice. É como ter uma lista de compras, onde cada item é numerado e você pode adicionar ou remover itens conforme necessário.

Digamos que queremos criar uma lista de tarefas a fazer durante o dia. Podemos fazer isso usando uma lista Python:



Listas - Dayana Ferreira.py

```
tarefas = ["Estudar Python", "Fazer exercícios", "Ler um livro"]  
print(tarefas[1]) # Saída: Fazer exercícios
```

Para acessar um item específico na lista, você usa seu índice correspondente. Lembre-se de que os índices começam em 0.

Uma das coisas legais sobre as listas é que você pode facilmente adicionar novos itens ou remover itens existentes.



Listas - Dayana Ferreira.py

```
tarefas.append("Fazer compras") # Adiciona um novo item ao final da lista  
print(tarefas)  
  
del tarefas[1] # Remove o item no índice 1  
print(tarefas)
```

# EXPLORANDO O PODER DAS LISTAS



As listas são incrivelmente versáteis e podem ser usadas em uma variedade de situações. Elas são fundamentais para organizar e manipular dados em Python.

● ● ● Listas - Dayana Ferreira.py

```
for tarefa in tarefas:  
    print(tarefa)
```

As listas podem conter uma mistura de diferentes tipos de dados. Por exemplo, você pode ter uma lista com números, strings e até mesmo outras listas!

● ● ● Listas - Dayana Ferreira.py

```
lista_mista = [10, "Python", True, [1, 2, 3]]
```

# CRIANDO LISTAS DE FORMA INTELIGENTE



A compreensão de lista é uma maneira elegante e concisa de criar listas em Python. Ela permite criar listas de forma mais eficiente e legível.

● ● ● Listas - Dayana Ferreira.py

```
numeros = [1, 2, 3, 4, 5]
quadrados = [x**2 for x in numeros]

# Resultado: quadrados = [1, 4, 9, 16, 25]
```

Python oferece uma variedade de operações para trabalhar com listas, como concatenação(+), fatiamento ([*inicio:fim*]), inversão (*reversed()*).

● ● ● Listas - Dayana Ferreira.py

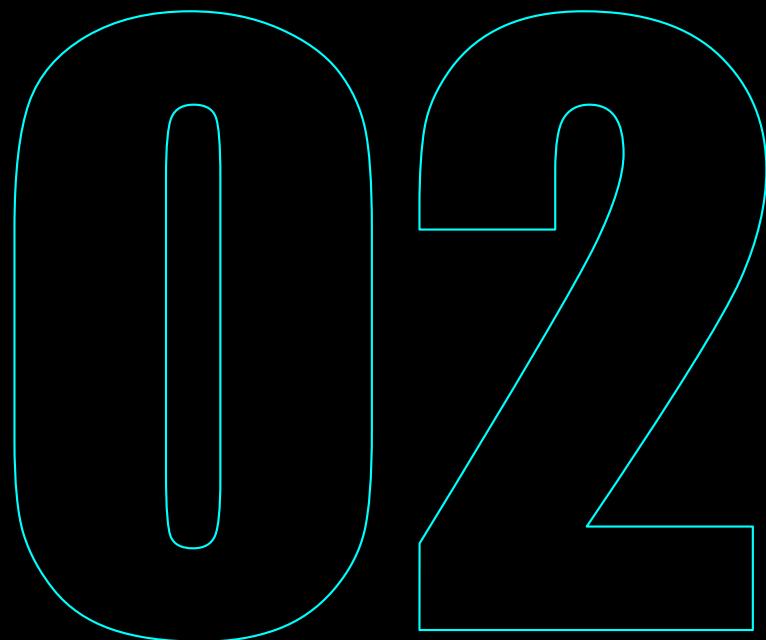
```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
concatenacao = lista1 + lista2

# Resultado: concatenacao = [1, 2, 3, 4, 5, 6]
```

Python possui várias funções embutidas úteis para trabalhar com listas, como *len()*, *max()*, *min()*, *sum()* e *sorted()*.

● ● ● Listas - Dayana Ferreira.py

```
numeros = [5, 2, 8, 1, 9]
tamanho = len(numeros) # Resultado: tamanho = 5
maior = max(numeros) # Resultado: maior = 9
soma = sum(numeros) # Resultado: soma = 25
```



# TUPLAS

---

## O GUARDIÃO DA INTEGRIDADE DOS DADOS

Quando se trata de proteger a integridade dos seus dados, as tuplas entram em cena como verdadeiros guardiões. Elas são como cofres onde você pode armazenar informações preciosas de forma segura e imutável.

# IMUTÁVEIS, ORDENADAS E CONFIÁVEIS



As tuplas são coleções ordenadas e imutáveis de itens. Isso significa que, uma vez criadas, elas não podem ser modificadas. É como ter um contrato selado - o conteúdo está lá para sempre, protegido contra qualquer alteração acidental.

Imagine que você está trabalhando em um projeto e precisa armazenar as coordenadas de um ponto no plano cartesiano. Você pode usar uma tupla para representar essas coordenadas:



Tuplas - Dayana Ferreira.py

```
ponto = (10, 20)

print(ponto[0]) # Saída: 10

ponto[0] = 15 # Isso resultará em um erro, pois as tuplas são imutáveis
```

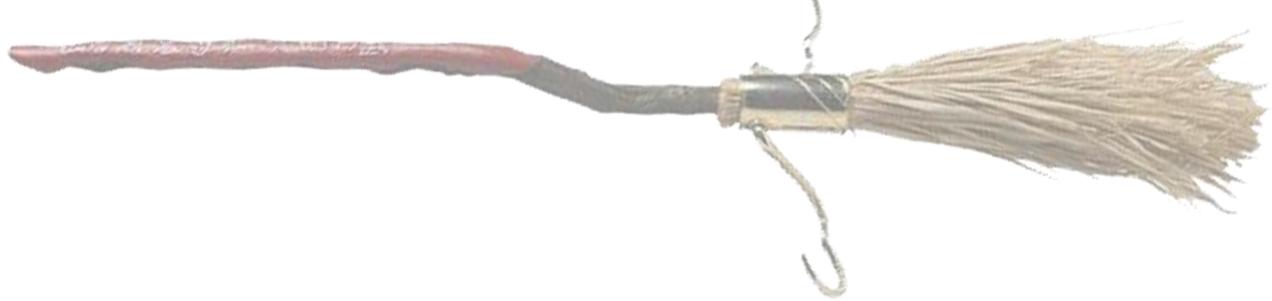
Assim como nas listas, você pode percorrer todos os itens em uma tupla usando um loop for.



Tuplas - Dayana Ferreira.py

```
for coordenada in ponto:
    print(coordenada)
```

# A PROFUNDIDADE DAS TUPLAS



Uma aplicação comum das tuplas é retornar múltiplos valores de uma função. Por exemplo, uma função pode retornar uma tupla contendo o resultado de um cálculo juntamente com um status de sucesso.



Tuplas - Dayana Ferreira.py

```
def dividir(dividendo, divisor):
    resultado = dividendo / divisor
    return resultado, "Sucesso" if divisor != 0 else "Erro"

resultado, status = dividir(10, 2)
print(resultado)      # Saída: 5.0
print(status)         # Saída: Sucesso
```

Como as tuplas são imutáveis, elas podem ser usadas como chaves em dicionários, ao contrário das listas. Isso ocorre porque as chaves de dicionários precisam ser hasháveis, ou seja, não podem ser modificadas após sua criação.



Tuplas - Dayana Ferreira.py

```
coordenadas = {(10, 20): "Casa", (30, 40): "Trabalho"}
print(coordenadas[(10, 20)]) # Saída: Casa
```

# CRIANDO TUPLAS DE FORMA INTELIGENTE



Você pode desempacotar os valores de uma tupla em variáveis individuais usando uma atribuição múltipla. Isso é útil quando você deseja acessar elementos individuais de uma tupla.

● ● ● Tuplas - Dayana Ferreira.py

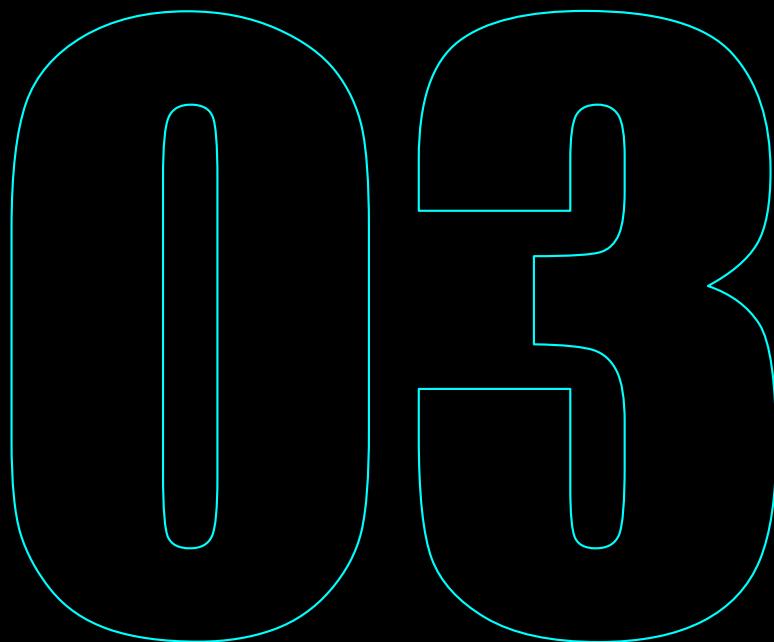
```
ponto = (5, 10)
x, y = ponto
print(x) # Saída: 5
print(y) # Saída: 10
```

O módulo `collections` em Python inclui uma estrutura de dados chamada "namedtuple", que é basicamente uma tupla com campos nomeados. Isso pode tornar o código mais legível e auto documentado.

● ● ● Tuplas - Dayana Ferreira.py

```
from collections import namedtuple

Ponto = namedtuple('Ponto', ['x', 'y'])
ponto = Ponto(5, 10)
print(ponto.x) # Saída: 5
print(ponto.y) # Saída: 10
```



# CONJUNTOS

---

DESCUBRA A MAGIA DA UNICIDADE

Os conjuntos são como varinhas mágicas que garantem que você nunca terá itens duplicados, permitindo que você explore a magia da unicidade em seus dados.

# DESCUBRA O PODER DA UNICIDADE



Os conjuntos em Python são coleções não ordenadas e mutáveis de elementos únicos. Eles agem como guardiões mágicos, permitindo apenas uma instância de cada elemento, eliminando duplicatas automaticamente.

Imagine que você tenha uma lista de números e queira remover duplicatas para obter uma lista de números únicos. Os conjuntos são perfeitos para isso:



Conjuntos - Dayana Ferreira.py

```
numeros = {1, 2, 3, 4, 5, 1, 2, 3}  
print(numeros) # Saída: {1, 2, 3, 4, 5}
```

Assim como as listas, os conjuntos também suportam compreensões, permitindo criar conjuntos de forma concisa e eficiente.



Conjuntos - Dayana Ferreira.py

```
numeros = {x for x in range(10) if x % 2 == 0} # Cria um conjunto com números pares de 0 a 9  
print(numeros) # Saída: {0, 2, 4, 6, 8}
```

# UNICIDADE E EFICIÊNCIA DE DADOS



Você pode adicionar novos elementos a um conjunto usando o método `add()` e remover elementos usando o método `remove()`.

```
● ○ ● Conjuntos - Dayana Ferreira.py

numeros.add(6) # Adiciona o número 6 ao conjunto
print(numeros) # Saída: {1, 2, 3, 4, 5, 6}

numeros.remove(3) # Remove o número 3 do conjunto
print(numeros) # Saída: {1, 2, 4, 5, 6}
```

Os conjuntos em Python oferecem uma variedade de operações poderosas, como união (`|`), interseção (`&`), diferença (`-`) e diferença simétrica (`^`), que permitem combinar e comparar conjuntos de forma eficiente.

```
● ○ ● Conjuntos - Dayana Ferreira.py

conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}

uniao = conjunto1 | conjunto2 # União de conjuntos
print(uniao) # Saída: {1, 2, 3, 4, 5}

intersecao = conjunto1 & conjunto2 # Interseção de conjuntos
print(intersecao) # Saída: {3}
```



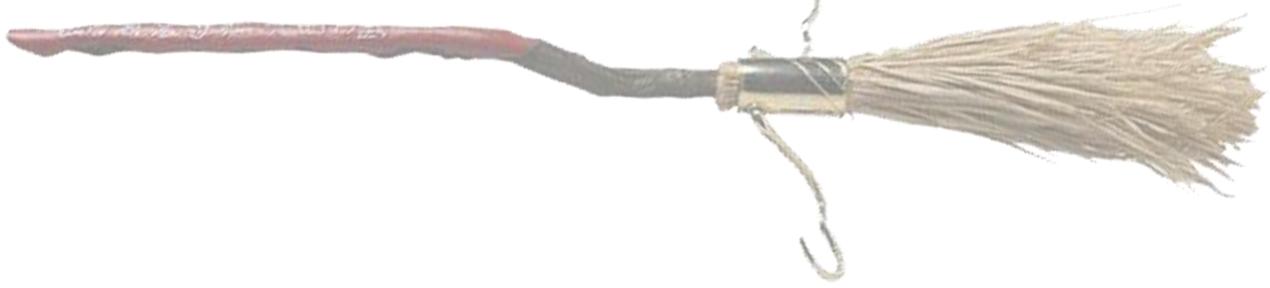
# DICIONÁRIOS

---

## A CHAVE PARA A ORGANIZAÇÃO EFICIENTE

A organização eficiente é alcançada por meio de chaves e valores. Os dicionários funcionam como enciclopédias, permitindo que você encontre informações rapidamente usando chaves únicas para acessar os valores correspondentes

# A CHAVE PARA A EFICIÊNCIA



Os dicionários são estruturas de dados que armazenam pares chave-valor, onde cada chave está associada a um valor específico. Eles são como índices em uma enciclopédia, onde você pode procurar por uma palavra-chave (chave) e encontrar a informação correspondente (valor) instantaneamente.

Suponha que você queira armazenar informações sobre uma pessoa, como nome, idade e cidade. Você pode fazer isso facilmente usando um dicionário:

```
● ● ● Dicionários - Dayana Ferreira.py  
  
pessoa = {'nome': 'Alice', 'idade': 25, 'cidade': 'São Paulo'}  
  
print(pessoa['idade']) # Saída: 25
```

Você pode adicionar novos pares chave-valor a um dicionário ou remover pares existentes conforme necessário.

```
● ● ● Dicionários - Dayana Ferreira.py  
  
pessoa['profissão'] = 'Engenheira' # Adiciona uma nova chave-valor ao dicionário  
del pessoa['cidade'] # Remove a chave 'cidade' do dicionário
```

# DESBRAVANDO OS DICIONÁRIOS



Você pode percorrer todos os pares chave-valor em um dicionário usando um loop for.

## ● ● ● Dicionários - Dayana Ferreira.py

```
for chave, valor in pessoa.items():
    print(chave + ':', valor)
```

Assim como as listas e conjuntos, os dicionários também suportam comprehensões, permitindo criar dicionários de forma concisa e eficiente.



## Dicionários - Dayana Ferreira.py

```
numeros = {x: x**2 for x in range(5)} # Cria um dicionário com os quadrados dos números de 0 a 4
print(numeros) # Saída: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Você pode criar dicionários que contêm outros dicionários como valores, permitindo uma estrutura de dados mais complexa e organizada.



## Dicionários - Dayana Ferreira.py

```
agenda = {'contatos': {'Alice': 123456, 'Bob': 789012}}
print(agenda['contatos']['Alice']) # Saída: 123456
```

# AGRADECIMENTOS

---

# QUE MASSA QUE VOCÊ CHEGOU ATÉ AQUI



Este eBook foi gerado por IA e diagramado por uma humana. Para o desafio de projeto do módulo 'Introdução à Engenharia de Prompts com ChatGPT' do curso 'Fundamentos de IA para Devs Santander 2024', você pode encontrar o passo a passo de como foi criado no [GitHub do autor](#). Aproveite a leitura e bons estudos!

Este conteúdo foi gerado com propósitos educacionais durante meu processo de aprendizado como desenvolvedora back end Python. Apesar da cuidadosa revisão humana, podem existir erros.  
Estou em constante evolução e aprendizado.

## Autora



Dayana Ferreira



[Github](#) | [linkedin](#) [Linkedin](#)

[Meu Perfil na DIO](#)