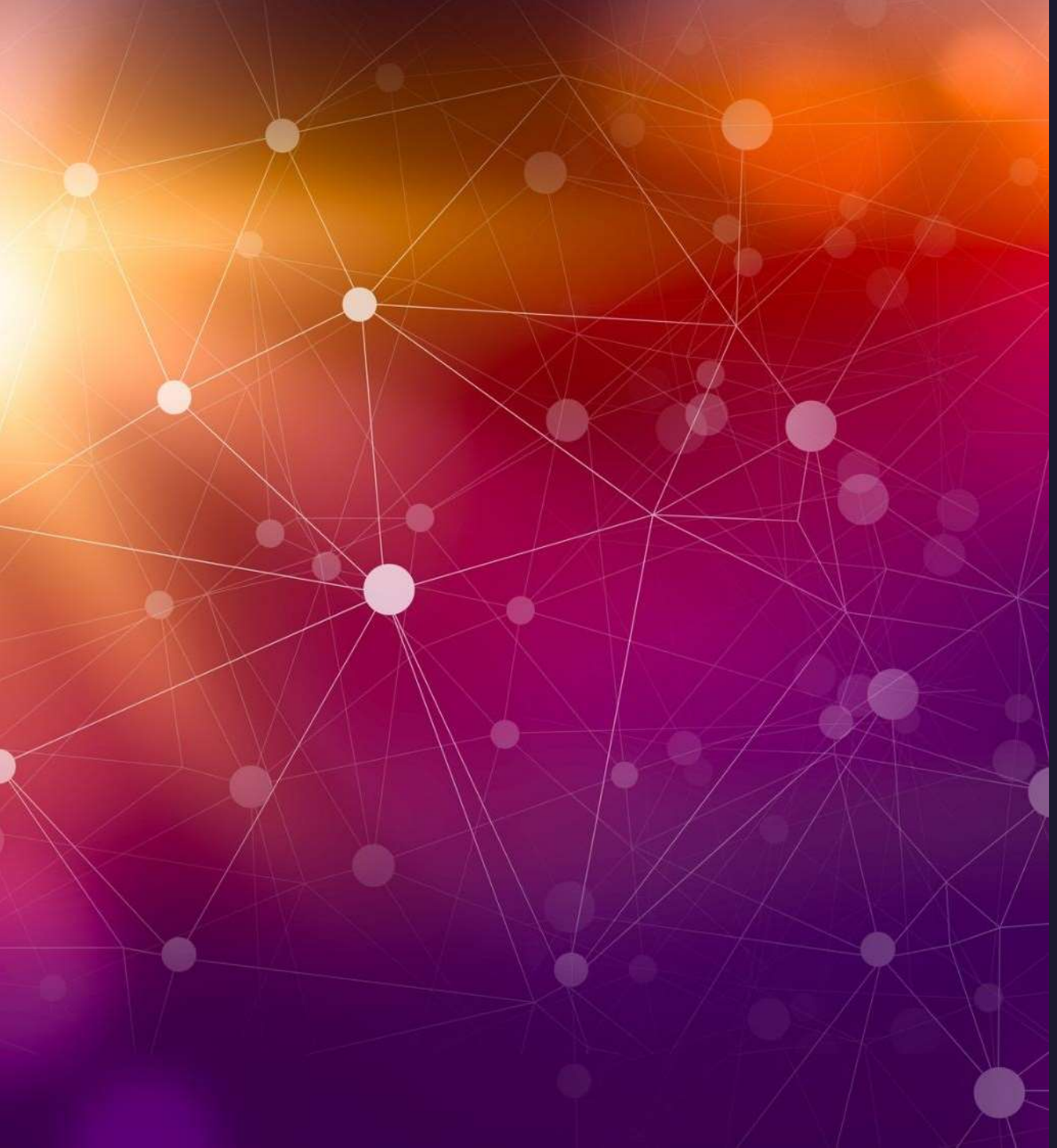
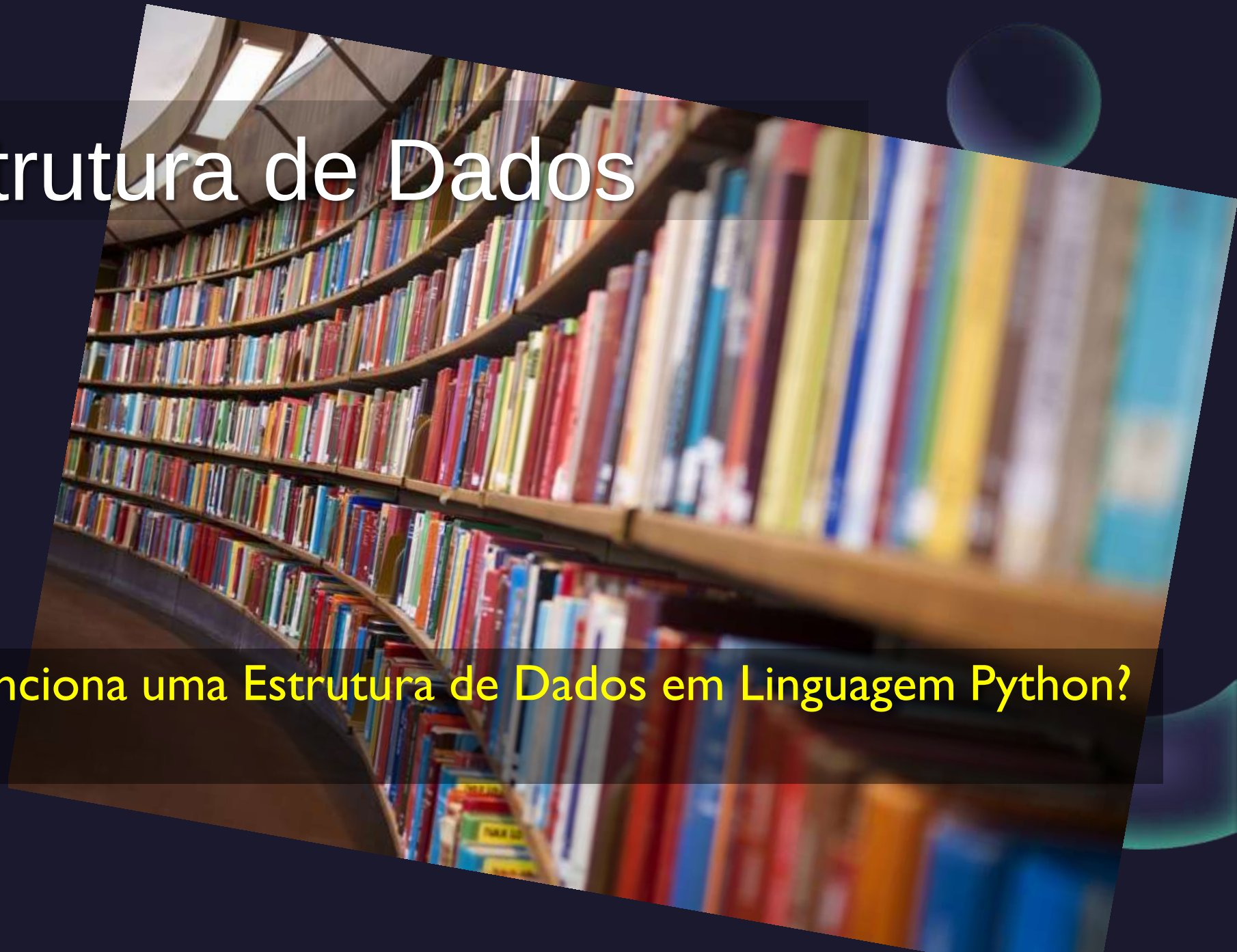


Linguagem de Programação Back-End

Prof. Wellington S. S. Silva

Python: Estrutura de Dados



Introdução: Como funciona uma Estrutura de Dados em Linguagem Python?

Estrutura de Dados– O que são e como funcionam

São formas de organizar e armazenar dados na memória para acesso e manipulação eficientes dessas informações.

Melhorando a performance do programa e dando mais robustez a aplicação que trará mais segurança e eficiência para o usuário final.

Coleção Fundamental de Estutura de Dados em Python

1. **Listas**
2. **Tuplas**
3. **Sets (Conjuntos)**
4. **Dicionário**

Coleção Fundamental de Estutura de Dados em Python

Listas

Estrutura de Dados - Listas

Uma **Lista** é uma estrutura de dados linear que armazena uma **sequência ordenada** de elementos do mesmo tipo. Pense em uma lista de compras, uma fila de banco ou os vagões de um trem: a característica principal é que existe uma ordem lógica entre os elementos.

Estrutura de Dados - Listas

Características Principais

Sequência Lógica: Os elementos estão organizados um após o outro. Mesmo que não estejam lado a lado na memória física do computador, eles têm uma relação de ordem (*predecessor* e *sucessor*).

Tamanho Variável (Geralmente): Diferentemente de um *Array* estático, uma lista pode **crescer ou diminuir** dinamicamente à medida que elementos são inseridos ou removidos (especialmente nas listas encadeadas).

Operações Comuns: As listas suportam operações básicas como:

- **Inserção:** Adicionar um novo elemento (no início, no fim ou em uma posição específica).
- **Remoção:** Excluir um elemento.
- **Busca (ou Acesso):** Encontrar ou acessar um elemento.

Estrutura de Dados - Listas

Tipos de Implementação de Listas

O modo como os dados são armazenados na memória define os principais tipos de listas:

1. Listas Sequenciais (ou Contíguas)

Como funciona: Os elementos são armazenados em **posições de memória fisicamente consecutivas** (em sequência).

Implementação: Geralmente são implementadas usando **Arrays (Vetores)**.

Vantagens:

- **Acesso Rápido (Acesso Direto/Constante):** Acessar um elemento por seu índice (x_i) é muito rápido, pois a posição de memória pode ser calculada diretamente.

Desvantagens:

- **Tamanho Fixo:** O tamanho máximo deve ser definido previamente (em algumas linguagens/contextos).
- **Movimentação de Dados:** Inserir ou remover um elemento no meio da lista é ineficiente, pois exige que todos os elementos seguintes sejam **movidos** para abrir espaço ou preencher a lacuna.

Estrutura de Dados - Listas

2. Listas Encadeadas (ou Ligadas)

Como funciona: Os elementos (**Nós**) podem estar espalhados pela memória. A ordem lógica é mantida através de **ponteiros** (ou referências). Cada **Nó** contém:

- O **Dado** (a informação).
- Um **Ponteiro** para o **próximo Nó** na sequência.

Vantagens:

- **Tamanho Dinâmico:** Crescem e diminuem facilmente, alocando memória apenas quando necessário.
- **Inserção/Remoção Eficiente:** Adicionar ou remover um Nó é rápido (basta reajustar os ponteiros), sem precisar mover os dados.

Desvantagens:

- **Acesso Sequencial:** Para acessar um elemento no meio da lista, é necessário **percorrer** a lista do início até ele (acesso lento).
- **Memória Extra:** Cada Nó precisa de espaço adicional para o ponteiro.

Estrutura de Dados - Listas

Variações de Listas Encadeadas

As listas encadeadas possuem variações importantes:

Tipo de Lista	Descrição
Simplesmente Encadeada →	Cada nó aponta apenas para o **próximo** elemento. A navegação é unidirecional.
Duplamente Encadeada →	Cada nó tem um ponteiro para o **próximo** e outro para o **anterior**. Permite navegação nos dois sentidos.
Circular →	O ponteiro do último elemento aponta de volta para o **primeiro** elemento.

Estrutura de Dados - Listas

Relação com Outras Estruturas

Listas são tão flexíveis que são usadas para implementar outras Estruturas de Dados Abstratas (ADTs):

Pilha (Stack): Uma lista linear onde inserção e remoção ocorrem apenas em uma ponta, chamada **Topo** **(LIFO - *Last-In, First-Out*).**

Fila (Queue): Uma lista linear onde a inserção ocorre em uma ponta (**Final**) e a remoção na outra (**Início**) **(FIFO - *First-In, First-Out*).**

Em resumo, as listas são cruciais para organizar dados de forma sequencial, sendo a escolha da implementação (sequencial ou encadeada) ditada pelos requisitos de velocidade de acesso e eficiência nas operações de inserção/remoção.

Estrutura de Dados - Listas

Coleção ordenada e mutável

- Permite elementos duplicados

Sintaxe: lista = [1, 2, 3]

Principais operações:
- append(), insert(), remove(), pop(), sort(), len()

Exemplo:
frutas = ['maçã', 'banana', 'laranja']

Estrutura de Dados - Listas

Resumo das Funções:

Função	O que faz	Exemplo
len()	Retorna o número de elementos na lista.	len(minha_lista)
append()	Adiciona um item **ao final** da lista.	minha_lista.append('novo')
insert()	Adiciona um item em uma **posição (índice)** específica.	minha_lista.insert(1, 'meio')
remove()	Remove a **primeira ocorrência** de um **valor** específico.	minha_lista.remove('erro')
pop()	Remove e retorna um item pelo seu **índice**.	item = minha_lista.pop(0)
sort()	**Ordena** a lista (crescente ou alfabética) **no local**.	minha_lista.sort()

Estrutura de Dados - Listas

Exemplos - Praticar

Exercícios de Treinamento em Listas (Python)

Exercício 1: Controle de Estoque (Adição e Verificação)

Objetivo: Praticar append() e len().

Um pequeno armazém precisa controlar seu inventário.

1. Crie uma lista chamada estoque e inicialize-a com os seguintes produtos: "Camisa", "Calça", "Meia".
2. Use o método append() para adicionar três novos produtos: "Sapato", "Cinto", e "Gravata".
3. Imprima a lista estoque completa.
4. Use a função len() para calcular e imprimir quantos produtos estão agora no estoque.

Exercício 2: Gerenciamento de Tarefas (Inserção e Remoção)

Objetivo: Praticar insert(), remove() e pop().

Uma lista de tarefas (to-do list) precisa ser ajustada durante o dia.

1. Crie a lista tarefas inicial: "Comprar Pão", "Estudar Python", "Pagar Contas".
2. Um item urgente surgiu: use insert() para adicionar "Responder Email Urgente" como a **primeira** tarefa (índice 0).
3. Você terminou de estudar Python: use remove() para tirar "Estudar Python" da lista.
4. Um item no final foi descartado: use pop() (sem índice) para remover o último item da lista e imprima qual foi o item descartado.
5. Imprima a lista tarefas final.

Exercício 3: Pódio e Classificação (Ordenação e Acesso)

Objetivo: Praticar sort(), acesso por índice e len().

Você tem uma lista desordenada de pontuações de atletas.

1. Crie uma lista chamada pontuacoes com os seguintes números: [45, 88, 70, 92, 65, 88].
2. Use o método sort() para ordenar a lista **em ordem decrescente** (do maior para o menor). *Dica: use o argumento reverse=True.*
3. Imprima a lista pontuacoes após a ordenação.
4. Use o acesso por índice ([]) para imprimir a **maior pontuação** (o 1º colocado) e a **terceira maior pontuação** (o 3º colocado).

Coleção Fundamental de Estutura de Dados em Python

Tuplas

Estrutura de Dados - Tuplas

Uma **tupla** é uma **coleção ordenada** e **imutável** de elementos.

Ou seja:

Mantém a **ordem** em que os elementos foram inseridos.

Não pode ser alterada após sua criação (não é possível adicionar, remover ou modificar elementos diretamente).

Em outras palavras: é como uma lista, mas **congelada**.

Estrutura de Dados - Tuplas

Características Principais

Ordenada: Os elementos são mantidos em uma ordem definida e têm um índice.

Heterogênea: Pode armazenar diferentes tipos de dados em seus elementos (números, strings, outras listas ou tuplas, etc.).

Imutável (Immutable): Esta é a característica ***chave***. Após a criação, você **não pode** alterar, adicionar ou remover elementos de uma tupla.

Sintaxe: Geralmente é definida usando **parênteses ()** e os elementos são separados por vírgulas. Ex.: (1, 2, 3)

Estrutura de Dados – Tuplas

Use tuplas quando:

Você tiver um conjunto de dados que **não deve mudar** durante a execução do programa (por exemplo, as cores primárias, as coordenadas GPS de um local fixo, ou as configurações iniciais).

Você precisar de uma **chave em um dicionário** (dict): como as tuplas são imutáveis, elas podem ser usadas como chaves, enquanto listas não podem.

Você quiser garantir a **integridade dos dados** de forma simples, evitando modificações acidentais.

As tuplas são essenciais para representar dados fixos e estruturados, oferecendo uma forma mais segura e, em certos casos, mais eficiente de armazenar coleções de dados.

Estrutura de Dados – Tuplas vs Listas

É muito comum confundir Tuplas e Listas, pois ambas armazenam coleções ordenadas. A distinção mais importante é a **mutabilidade:**

Característica	Tupla (tuple)	Lista (list)
Sintaxe	Usam parênteses: (1, 2, 3)	Usam colchetes: [1, 2, 3]
Mutabilidade	**Imutável** (Não pode ser alterada)	**Mutável** (Pode ser alterada)
Uso Comum	Dados fixos, coordenadas (x, y), chaves de dicionário.	Coleções que mudam (estoque, tarefas, filas).
Performance	Geralmente mais **rápida** e ocupa menos memória.	Geralmente mais lenta em algumas operações.

Estrutura de Dados – Tuplas

Algumas da possibilidades de utilização para as **Tuplas**:

1. Criação e Acesso

Você acessa os elementos pelo índice, exatamente como em uma lista:

```
coordenada = (10, 25)
print(coordenada[0])                # A Saída: 10
```

Você também pode criar uma tupla sem parênteses, apenas com vírgulas (O Python infere)

```
dados_pessoais = "Alice", 30, "Engenheira"
print(dados_pessoais)               # A Saída será: ('Alice', 30, 'Engenheira')
```

Estrutura de Dados – Tuplas

Algumas da possibilidades de utilização para as **Tuplas**:

2. Desempacotamento (*Unpacking*)

Uma das utilizações mais elegantes das tuplas é o **desempacotamento**, onde você atribui os valores da tupla a múltiplas variáveis em uma única linha:

```
ponto = (3.5, 7.2, 1.0)

x, y, z = ponto                 # O número de variáveis deve ser igual ao número de elementos da tupla


print(f"Valor de X: {x}")       # Saída: Valor de X: 3.5

print(f"Valor de Y: {y}")       # Saída: Valor de Y: 7.2
```

Estrutura de Dados – Tuplas

Algumas da possibilidades de utilização para as **Tuplas**:

3. Métodos Específicos

As tuplas possuem apenas dois métodos embutidos:

count(): Retorna o número de vezes que um valor aparece na tupla.

index(): Retorna o índice da primeira ocorrência de um valor.

Exemplo de Métodos Específicos em Tuplas

Vamos usar uma tupla chamada **notas_finais** que contém as notas de uma turma. Note que a nota 8.0 aparece mais de uma vez.

Estrutura de Dados – Tuplas

Exemplo de Métodos Específicos em Tuplas

```
# 1. Criação da Tupla (Imutável)

notas_finais = (10.0, 7.5, 8.0, 6.0, 9.5, 8.0, 5.0)

print(f"Tupla de Notas: {notas_finais}")

print("-" * 30)

# Queremos saber quantas vezes a nota 8.0 aparece.

contagem_8 = notas_finais.count(8.0)

print(f"Usando count(): A nota 8.0 aparece {contagem_8} vez(es).")

# Queremos saber onde a nota 9.5 está localizada.

indice_95 = notas_finais.index(9.5)

print(f"Usando index(): A nota 9.5 está no índice {indice_95}.")

#Vamos encontrar o índice da nota 8.0 (Ele retorna apenas o primeiro índice que encontra)

primeiro_indice_8 = notas_finais.index(8.0)

print(f"Usando index(): A primeira ocorrência de 8.0 está no índice {primeiro_indice_8}.")
```

Estrutura de Dados – Tuplas

Resumo das Saídas das **Tuplas:**

Método	Resultado	Explicação
count(8.0)	**2**	O valor 8.0 está presente duas vezes na tupla.
index(9.5)	**4**	O valor 9.5 é o 5º elemento, portanto, tem o índice 4.
index(8.0)	**2**	O valor 8.0 aparece primeiro no índice 2, e o método para de procurar ali.

Estrutura de Dados – Tuplas

Exercícios de Treinamento em Tuplas (Python)

Exercício 1: Dados Fixos e Acesso Rápido

Objetivo: Praticar a **criação** de tuplas e o **acesso por índice** (`[]`).

Crie uma tupla para armazenar os dados de um planeta que não devem mudar: nome, raio e densidade.

1. Crie uma tupla chamada `dados_planeta` com os seguintes valores: `"Marte"`, `3389.5` (raio em km) e `3.93` (densidade em g/cm³).

2. Imprima o **nome** do planeta usando o índice correto.

3. Imprima a **densidade** do planeta usando o índice correto.

4. Use o índice `-1` para imprimir o **último** elemento da tupla.

Estrutura de Dados – Tuplas

Exercícios de Treinamento em Tuplas (Python)

Exercício 2: Contagem e Localização de Elementos

Objetivo: Praticar os métodos de leitura `count()` e `index()`.

Em um registro de votação, alguns números de chapa apareceram repetidos.

1. Crie uma tupla chamada `votos` com os números: `(101, 205, 101, 300, 205, 101, 400)`.

2. Use o método `count()` para descobrir e imprimir quantas vezes a chapa `101` foi registrada.

3. Use o método `index()` para descobrir e imprimir a **primeira** posição (índice) onde o voto da chapa `300` aparece.

4. Tente usar o `append()` ou o acesso por índice para **adicionar** um novo voto (`500`) e observe o erro (`AttributeError` ou `TypeError`). *Seus alunos devem explicar por que isso acontece.*

Estrutura de Dados – Tuplas

Exercícios de Treinamento em Tuplas (Python)

Exercício 3: Desempacotamento de Dados

Objetivo: Praticar o **desempacotamento** (*unpacking*).

Uma função (que não precisamos criar) retorna as dimensões de uma caixa em uma única tupla.

1. Crie uma tupla chamada `dimensoes` com os valores: `(2.5, 5.0, 1.2)` (comprimento, largura, altura).

2. Use o **desempacotamento** para atribuir cada valor da tupla a três variáveis distintas, chamadas: `comprimento`, `largura` e `altura`.

3. Calcule e imprima a **área da base** da caixa, que é `comprimento * largura`.

4. Calcule e imprima o **volume** da caixa, que é `comprimento * largura * altura`.

Coleção Fundamental de Estutura de Dados em Python

Sets (Conjuntos)

Estrutura de Dados – Sets (Conjuntos)

Sets (Conjuntos) em Estruturas de Dados

Os **Sets**, ou **Conjuntos**, são uma das quatro estruturas de dados básicas que são as Listas, as Tuplas, os Sets (Conjuntos) e o Dicionário. Eles são implementados em várias linguagens de programação, como Python e Java, e se baseiam no conceito matemático de conjuntos.

Estrutura de Dados – Sets (Conjuntos)

Características Principais

Um **Set** (Conjunto) possui três características fundamentais que o diferenciam das outras estruturas:

Não-Ordenados: Os elementos em um Set (Conjunto) **não são armazenados em uma ordem específica**. Isso significa que a ordem em que você insere os elementos não é garantida quando você os acessa ou itera sobre o conjunto.

Mutáveis (na maioria das implementações): O Set (Conjunto) em si é **mutável**, o que significa que você pode **adicionar** ou **remover** elementos dele após sua criação. No entanto, os **elementos** armazenados dentro do conjunto devem ser **imutáveis** (como números, strings ou tuplas). Você **não pode** ter uma lista (que é mutável) como elemento de um Set (Conjunto).

Elementos Únicos: Esta é a **característica mais importante**. Um Set **não pode conter elementos duplicados**. Se você tentar adicionar um elemento que já existe no conjunto, o conjunto simplesmente o ignorará.

Estrutura de Dados – Sets (Conjuntos)

Casos de Uso Comuns

Os Sets (Conjunto) são extremamente úteis quando você precisa:

Remover Duplicatas: O uso mais comum é para **eliminar rapidamente elementos duplicados** de uma lista ou de uma coleção. Basta converter a coleção para um Set (Conjunto) e, em seguida, voltar para a estrutura de sua preferência.

Testes de Pertencimento Rápidos: Como a busca é muito eficiente, verificar se um elemento está presente em um grande conjunto de dados é uma opção utilizar essa operação que é muito rápida.

Realizar Operações de Conjunto: Eles são ideais para realizar operações matemáticas de conjuntos, como:

- **União:** Combinar todos os elementos de dois ou mais Sets (Conjuntos).
- **Interseção:** Encontrar os elementos que são comuns a dois ou mais Sets (Conjuntos).
- **Diferença:** Encontrar os elementos que estão em um Set (Conjunto), mas não em outro.
- **Diferença Simétrica:** Encontrar os elementos que estão em um dos Sets (Conjuntos), mas não em ambos.

Estrutura de Dados – Sets (Conjuntos)

Exemplo em Python

Em Python, um Set (Conjunto) é definido usando chaves { } ou a função set( ):

```
# Criação de um Set

frutas = {"maçã", "banana", "laranja", "maçã"}

# A saída será {'banana', 'laranja', 'maçã'} – veja que a duplicata 'maçã' foi removida.



# Adicionando um elemento

frutas.add("uva")

# A saída será {'banana', 'laranja', 'maçã', 'uva'} – veja que o item ‘uva' foi adicionado.
```

Estrutura de Dados – Sets (Conjuntos)

Exemplo em Python

Em Python, um Set (Conjunto) é definido usando chaves { } ou a função set( ):

```
# Tentativa de adicionar um elemento duplicado

frutas.add("banana")

# A saída será {'banana', 'laranja', 'maçã', 'uva'} – veja que a inserção do item 'banana' duplicado foi ignorado.



# Operação de Interseção

frutas_doces = {"uva", "manga", "banana"}

comuns = frutas.intersection(frutas_doces)

# A saída da variável "comuns" será {'banana', 'uva'}
```

Estrutura de Dados – Sets (Conjuntos)

Exemplo em Python

1. Removendo Duplicatas (Unicidade)

```
# Lista com valores duplicados (Lista é mutável e aceita repetições)

minha_lista_baguncada = [10, 20, 30, 10, 40, 20, 50, 30]

# Passo 1: Converter a lista para um Set

# O Set remove automaticamente os elementos duplicados

conjunto_sem_duplicatas = set(minha_lista_baguncada)


# Passo 2 (Opcional): Converter o Set de volta para uma Lista, se necessário

lista_limpa = list(conjunto_sem_duplicatas)

print(f"Lista Original: {minha_lista_baguncada}")

print(f"Set (Sem Duplicatas): {conjunto_sem_duplicatas}")

print(f"Lista Limpa: {lista_limpa}")
```

```
# Output:

# Lista Original: [10, 20, 30, 10, 40, 20, 50, 30]

# Set (Sem Duplicatas): {50, 20, 40, 10, 30}

# Note a ordem que pode mudar, pois Sets não são ordenados!
```

Estrutura de Dados – Sets (Conjuntos)

Exemplo em Python

2. Teste de Pertencimento Rápido (Performance)

```
# Criando um Set de usuários online

usuarios_online = {"Ana", "Bruno", "Carla", "Daniel", "Elias", "Fernanda"}

 # Nome a ser verificado

nome_buscado = "Bruno"

outro_nome = "Gabriel"

 #Verificação de pertencimento usando o operador 'in'

esta_online = nome_buscado in usuarios_online

nao_esta_online = outro_nome in usuarios_online

print(f"O usuário '{nome_buscado}' está online? {'Sim' if esta_online else 'Não'}")

print(f"O usuário '{outro_nome}' está online? {'Sim' if nao_esta_online else 'Não'}")

# Adicionando um novo usuário

usuarios_online.add("Gabriel")

print(f"Novo Set após adição: {usuarios_online}")
```

```
# Output:

# O usuário 'Bruno' está online? Sim

# O usuário 'Gabriel' está online? Não

# Novo Set após adição: {'Ana', 'Bruno', 'Carla', 'Daniel', 'Elias', 'Fernanda', 'Gabriel'}
```

Estrutura de Dados – Sets (Conjuntos)

Exemplo em Python

3. Operações Matemáticas de Conjunto (Interseção e União)

```
# Conjunto 1: Clientes que compraram Livros

compraram_livros = {"Alice", "Bob", "Charlie", "David"}

 # Conjunto 2: Clientes que compraram E-books (digital)

compraram_ebooks = {"Alice", "Charlie", "Eve", "Frank"}

 # Interseção (Elementos em COMUM) - Clientes que compraram Livros E E-books

ambos = compraram_livros.intersection(compraram_ebooks)

 # União (TODOS os elementos, sem duplicatas) - Clientes que compraram Livros OU E-books

qualquer_compra = compraram_livros.union(compraram_ebooks)

print(f"Clientes que compraram Livros e E-books (Interseção): {ambos}")

print(f"Clientes que fizeram qualquer compra (União): {qualquer_compra}")
```

```
# Output:

# Clientes que compraram Livros e E-books (Interseção): {'Alice', 'Charlie’}

# Clientes que fizeram qualquer compra (União): {'Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'}
```

Estrutura de Dados – Sets (Conjuntos)

Exercício 1: Limpeza e Contagem (Unicidade)

Objetivo: Usar a característica de unicidade do Set para remover duplicatas e descobrir quantos elementos únicos existem em uma lista.

Enunciado:
Dada a seguinte lista de números que representam as notas tiradas por uma turma em um teste, encontre:

Quantas notas **únicas** (sem repetição) existem na lista.

Quais são essas notas únicas.

```
# Dados de entrada
notas_turma = [7.5, 8.0, 6.5, 9.0, 7.5, 8.0, 5.0, 9.0, 10.0, 6.5]
```

Estrutura de Dados – Sets (Conjuntos)

Exercício 2: O Desafio de Palavras Comuns (Interseção)

Objetivo: Utilizar a operação de **interseção** para identificar elementos que estão presentes em ambos os conjuntos.

Enunciado:
Temos dois conjuntos de palavras. O **Set A** contém palavras que são classificadas como "Substantivos" e o **Set B** contém palavras classificadas como "Adjetivos".

Encontre:
Um novo conjunto contendo as palavras que, por engano, foram classificadas em **ambos** os conjuntos.

```
# Dados de entrada
set_a_substantivos = {"casa", "azul", "carro", "bonito", "cidade", "rápido"}

set_b_adjetivos = {"grande", "rápido", "azul", "feliz", "carro", "pequeno"}
```

Estrutura de Dados – Sets (Conjuntos)

Exercício 3: Verificação de Acesso (Pertencimento e Adição)

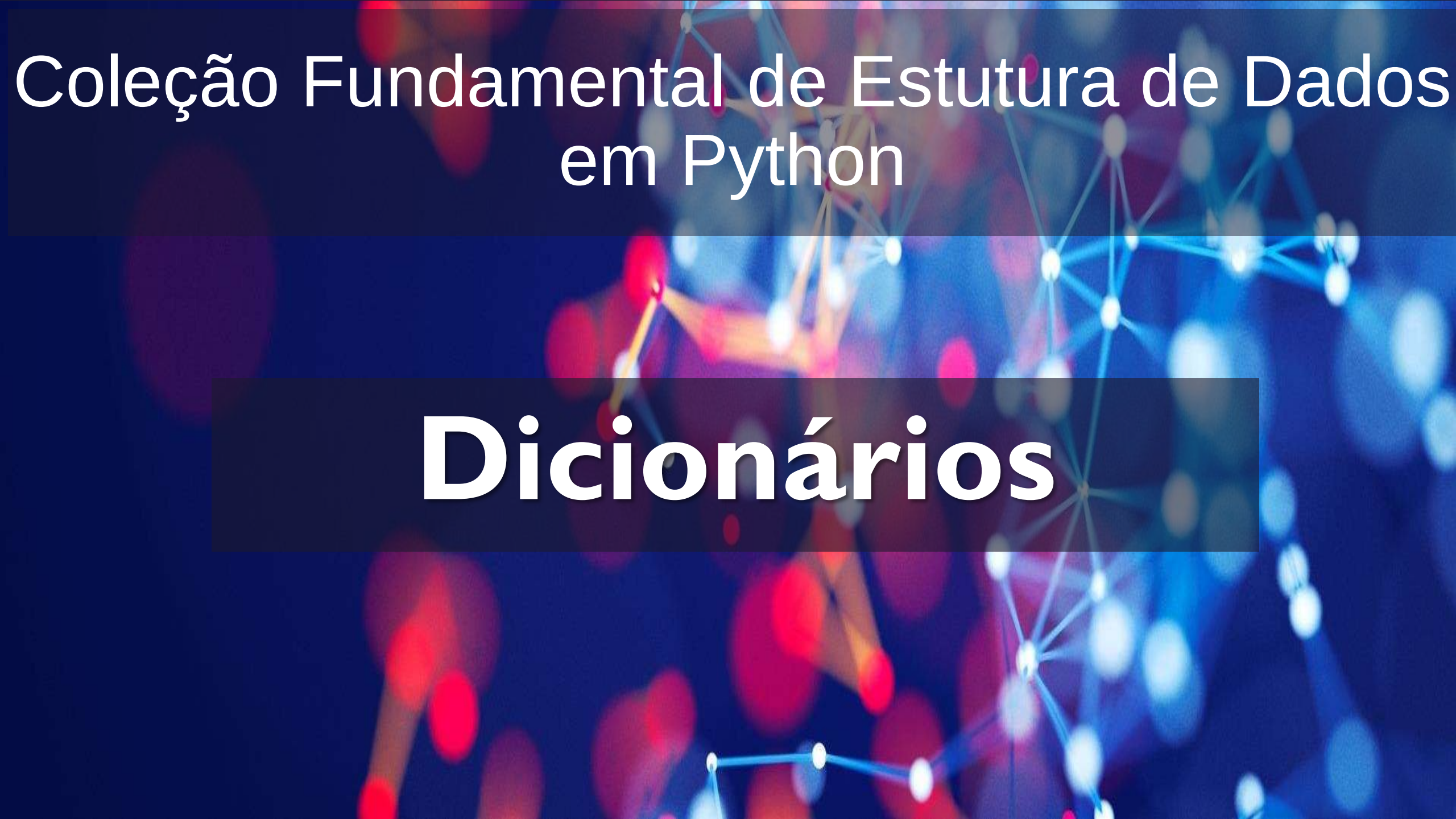
Objetivo: Praticar a verificação de pertencimento (in) e a mutabilidade do Set com as operações de adicionar e remover.

Enunciado:
Temos um conjunto que rastreia os "Códigos de Acesso Permitidos".

Verifique se o código **C-105** está atualmente na lista de acessos permitidos.
Se o código **C-105 não** estiver permitido, adicione-o ao conjunto.
Verifique se o código **A-201** está permitido e, se estiver, remova-o do conjunto por razões de segurança.

```
# Dados de entrada

codigos_permitidos = {"A-201", "B-300", "D-450", "E-700"}
```


Coleção Fundamental de Estutura de Dados em Python
Dicionários

Estrutura de Dados - Dicionários

Se as Listas são como uma "lista de compras" (ordenada e acessada por posição), os **Dicionários** são como uma **agenda telefônica** ou, literalmente, um dicionário de idiomas.

O Conceito Principal: Chave e Valor (Key-Value)

A característica fundamental do Dicionário é que ele **não** usa índices numéricos (0, 1, 2...) para guardar dados. Em vez disso, ele usa um sistema de **Mapeamento**.

Para cada informação que você guarda, você precisa de duas coisas:

Chave (Key): O "nome" único que identifica aquele dado (ex: "Nome", "Idade", "CPF").

Valor (Value): O conteúdo armazenado associado àquela chave.

Estrutura de Dados - Dicionários

Analogia: Imagine um armário de guarda-volumes.

Na **Lista**, você precisa saber que seu pertence está na gaveta número 3.

No **Dicionário**, a gaveta tem uma etiqueta escrita "Pertences do João". Você procura pela etiqueta (Chave) para pegar o conteúdo (Valor).

Características dos Dicionários

Sintaxe: Utilizam chaves { } para serem criados.

Chaves Únicas: Você não pode ter duas chaves iguais no mesmo dicionário (ex: dois campos "Nome"). Se tentar, o segundo valor substituirá o primeiro.

Mutáveis: Você pode adicionar novos campos, remover campos ou alterar valores a qualquer momento.

Tipagem Dinâmica: O *Valor* pode ser qualquer coisa (um número, uma string, uma lista inteira ou até outro dicionário). A *Chave* geralmente é uma String ou Número (precisa ser imutável).

Estrutura de Dados - Dicionários

Exemplo Prático

```
# Criando um dicionário
aluno = {
    "nome": "Carlos Silva",
    "idade": 25,
    "curso": "Análise de Sistemas",
    "notas": [8.5, 9.0, 7.5],  # O valor pode ser uma Lista!
    "ativo": True
}
# 1. Acessando valores pela Chave
print(aluno["nome"])
                    # Saída: Carlos Silva
```

```
# 2. Adicionando um novo par Chave-Valor
aluno["semestre"] = 4

# 3. Alterando um valor existente
aluno["curso"] = "Engenharia de Software“

# 4. Acessando as chaves e valores separadamente
print(aluno.keys())
                    # Mostra: nome, idade, curso, notas, ativo, semestre
print(aluno.values())
                    # Mostra os conteúdos
```

Estrutura de Dados - Dicionários

Principais Métodos

O Python oferece ferramentas poderosas para manipular dicionários:

Método	Função
.get('chave')	Tenta pegar um valor. Se a chave não existir, ele não dá erro (retorna None), o que é mais seguro que usar [].
.pop('chave')	Remove o item com aquela chave e retorna o valor removido.
.update({...})	Adiciona vários itens de uma vez ou atualiza os existentes.
.items()	Retorna uma lista de tuplas contendo (chave, valor), muito útil para loops for.

Estrutura de Dados - Dicionários

Quando usar Dicionários em vez de Listas?

Use **Listas** quando a ordem importa e você tem uma coleção de itens similares (ex: nomes de todos os alunos da sala).

Use **Dicionários** quando você tem dados estruturados ou precisa de "rótulos" para os dados (ex: todos os detalhes de *um único* aluno).

Estrutura de Dados - Dicionários

```
# Uma LISTA contendo DICIONÁRIOS
estoque = [
    {
        "id": 1,
        "produto": "Notebook Gamer",
        "preco": 4500.00,
        "quantidade": 5
    },
    {
        "id": 2,
        "produto": "Mouse Sem Fio",
        "preco": 120.50,
        "quantidade": 30
    },
    {
        "id": 3,
        "produto": "Teclado Mecânico",
        "preco": 350.00,
        "quantidade": 12
    }
]
print(estoque[0]["produto"])
                # Saída: Notebook Gamer
```

Estrutura de Dados - Dicionários

Insira o código abaixo no final do código anterior

```
print("--- RELATÓRIO DE ESTOQUE ---")

for item in estoque:

    # 'item' se torna o dicionário da vez a cada rodada do loop

    nome = item["produto"]

    valor = item["preco"]

    print(f"O produto {nome} custa R$ {valor}")
```

Estrutura de Dados - Dicionários

Utilizando o método .get

```
# Criando um dicionário de produtos e preços
produtos = {
    "Notebook": 3500.00,
    "Mouse": 50.00,
    "Teclado": 120.00
}

# Cenario 1: Buscando um item que EXISTE
# Funciona igual ao acesso normal
preco_mouse = produtos.get("Mouse")
print(f"Preço do Mouse: {preco_mouse}")

# Cenario 2: Buscando um item que NÃO EXISTE (Sem valor padrão)
# O retorno será 'None' (vazio), mas o programa continua rodando
preco_monitor = produtos.get("Monitor")
print(f"Preço do Monitor: {preco_monitor}")

# Cenario 3: Buscando um item que NÃO EXISTE (Com valor padrão)
# Aqui definimos o que retornar caso a chave falhe. Muito útil para interfaces.
preco_webcam = produtos.get("Webcam", "Produto indisponível")
print(f"Preço da Webcam: {preco_webcam}")
```

Estrutura de Dados - Dicionários

Utilizando o método .pop

```
# Criando um dicionário de tarefas
tarefas = {
    "segunda": "Estudar Python",
    "terca": "Reunião de equipe",
    "quarta": "Academia"
}

print(f"Lista original: {tarefas}")

# 1. Removendo um item que EXISTE
# O .pop remove a chave 'segunda' e guarda o valor na variável 'tarefa_feita'

tarefa_feita = tarefas.pop("segunda")

print(f"Tarefa concluída e removida: {tarefa_feita}")
print(f"Lista atualizada: {tarefas}")
```

```
# 2. Usando .pop com valor padrão (Segurança)

# Se tentarmos remover algo que não existe sem o valor padrão, o Python gera erro (KeyError).

# Com o valor padrão (segundo argumento), ele retorna o aviso e o código não quebra.

tarefa_quinta = tarefas.pop("quinta", "Não há tarefa para este dia")

print(f"Tentativa de remoção: {tarefa_quinta}")
```

Estrutura de Dados - Dicionários

Utilizando o método .update

```
# 1. Dicionário inicial (Configuração atual)
configuracao = {
    "usuario": "admin",
    "tema": "claro",
    "volume": 50
}
print(f"Antes: {configuracao}")

# 2. Novos dados que chegaram (ex: o usuário mudou as opções)
# Note que 'tema' é uma chave repetida e 'idioma' é nova
novas_opcoes = {
    "tema": "escuro",       #Vai atualizar o existente
    "idioma": "PT-BR"       #Vai ser adicionado
}

# 3.Aplicando o update
# O dicionário 'configuracao' absorve os dados de 'novas_opcoes'
configuracao.update(novas_opcoes)

print(f"Depois: {configuracao}")
```

Estrutura de Dados - Dicionários

Utilizando o método .items

```
# Dicionário de notas dos alunos
boletim = {
    "Ana": 9.5,
    "Bruno": 7.0,
    "Carlos": 4.5,
    "Daniela": 10.0
}

# Usando .items() para pegar Nome e Nota ao mesmo tempo
# 'aluno' recebe a chave, 'nota' recebe o valor
print("--- RESULTADOS FINAIS ---")

for aluno, nota in boletim.items():
    # Podemos usar lógica com os valores dentro do loop
    status = "Aprovado" if nota >= 6 else "Reprovado"

    print(f"Aluno: {aluno} | Nota: {nota} | Status: {status}")
```

Estrutura de Dados - Dicionários

1° Exercício, utilizando os 4 métodos

Exercício: O Carrinho de Compras da Black Friday

Cenário: O sistema recebeu o carrinho de compras de um cliente e precisa processar o pedido final.

Adicionar Frete (update): O cliente calculou o frete e ele precisa entrar no pedido.

Verificar Cupom (get): O sistema verifica se o cliente digitou um cupom de desconto (pode não ter digitado).

Remover Item (pop): O cliente desistiu de um produto de última hora porque estava caro.

Gerar Nota Fiscal (items): Listar tudo o que sobrou e calcular o total.

Estrutura de Dados - Dicionários

2° Exercício, utilizando os 4 métodos

Exercício: O Perfil do Influenciador Digital

Cenário: O aluno deve gerenciar o back-end de um perfil de rede social.

Atualizar Bio (update): O usuário mudou a descrição e adicionou um site.

Checar Selo de Verificado (get): O sistema precisa saber se exibe o ícone azul (muitos não têm).

Deletar Rascunho (pop): O usuário descartou uma postagem que estava pendente.

Exibir Dashboard (items): Mostrar as métricas do perfil.

Estrutura de Dados - Dicionários

3° Exercício, utilizando os 4 métodos

Exercício: O Sistema de RH (Folha de Pagamento)

Cenário: Processamento de dados de um funcionário no dia do pagamento.

Bônus e Promoção (update): O funcionário foi promovido e o salário mudou.

Benefício Transporte (get): Verificar se ele optou pelo Vale Transporte (nem todos optam).

Quitar Empréstimo (pop): O funcionário pagou a última parcela do empréstimo consignado, então a dívida sai do sistema.

Relatório Financeiro (items): Listar os dados finais para o contador.

Estrutura de Dados - Dicionários

Exercício, utilizando as 4 Estruturas de Dados

Enunciado do Exercício

Um pequeno e-commerce registrou um novo pedido. Os dados brutos do pedido são os seguintes:

Itens Comprados (Lista): Uma lista de produtos que o cliente colocou no carrinho, que pode conter itens repetidos.

Detalhes do Cliente (Tupla): Informações do cliente que não devem ser alteradas (imutáveis).

Preços dos Produtos (Dicionário): O preço de cada produto único.

```
# Dados de entrada
itens_comprados = ["camiseta", "caneca", "camiseta", "chaveiro", "caneca", "meia"]
detalhes_cliente = ("maria.silva@email.com", "Rua das Flores, 123", "SP")
precos_produtos = {"camiseta": 35.00, "caneca": 25.00, "chaveiro": 10.00, "meia": 15.00}
```

Estrutura de Dados - Dicionários

Exercício, utilizando as 4 Estruturas de Dados

Tarefas:

Sets (Conjuntos): Use um Set para descobrir quais são os **produtos únicos** que o cliente comprou.

Dicionário: Calcule o **valor total** do pedido. Para isso, você precisará percorrer a lista original de itens_comprados e usar o precos_produtos para somar o valor.

Lista: Crie uma nova lista chamada pedido_final que armazene cada item comprado junto com seu preço, no formato de tupla (nome_do_item, preco_do_item).

Tupla: Exiba o nome do cliente (a primeira parte do email antes do '@') usando a detalhes_cliente (Tupla).

Obrigado

Wellington S. S. Silva

guitomw@outlook.com