

Aula 2: Sets e Dicionários

Desvendando estruturas de dados poderosas em Python

Objetivo da Aula

Nesta aula, vamos mergulhar em duas estruturas de dados fundamentais em Python: os **dicionários** e os **sets**.



Mapeamento de Dados

Compreender como os dicionários funcionam como "**mapas**" para organizar informações com pares de **chave-valor**.



Conjuntos Únicos

Explorar o conceito de **sets** para lidar com coleções de elementos sem duplicatas e sem ordem definida.



Aplicações Práticas

Aprender a manipular ambas as estruturas e aplicá-las na resolução de problemas do dia a dia da programação.

Sets: Os Conjuntos de Elementos Únicos



Sets são coleções **não ordenadas** e que armazenam apenas **elementos únicos**. Se você tentar adicionar um item que já existe, ele será ignorado.

Eles são perfeitos para operações matemáticas de conjunto, como união, interseção e diferença, além de serem muito rápidos para verificar a existência de um elemento.

```
frutas = set() # set vazio  
  
frutas = {"maçã", "banana", "uva", "maçã"}  
  
print(frutas) # {'maçã', 'banana', 'uva'}
```

Repare que a segunda "maçã" foi automaticamente removida ao criar o set.

❓ Atividade set:

Crie um set vazio chamado `set_compras` e adicione itens dentro dele. Ao final exiba este set.

Dica: Utilize o método `.add()` para adicionar itens ao set.

Introdução aos Dicionários

Dicionários são coleções de dados que armazenam informações em pares de **chave: valor**. Pense neles como um guia telefônico, onde o nome é a chave e o número é o valor.

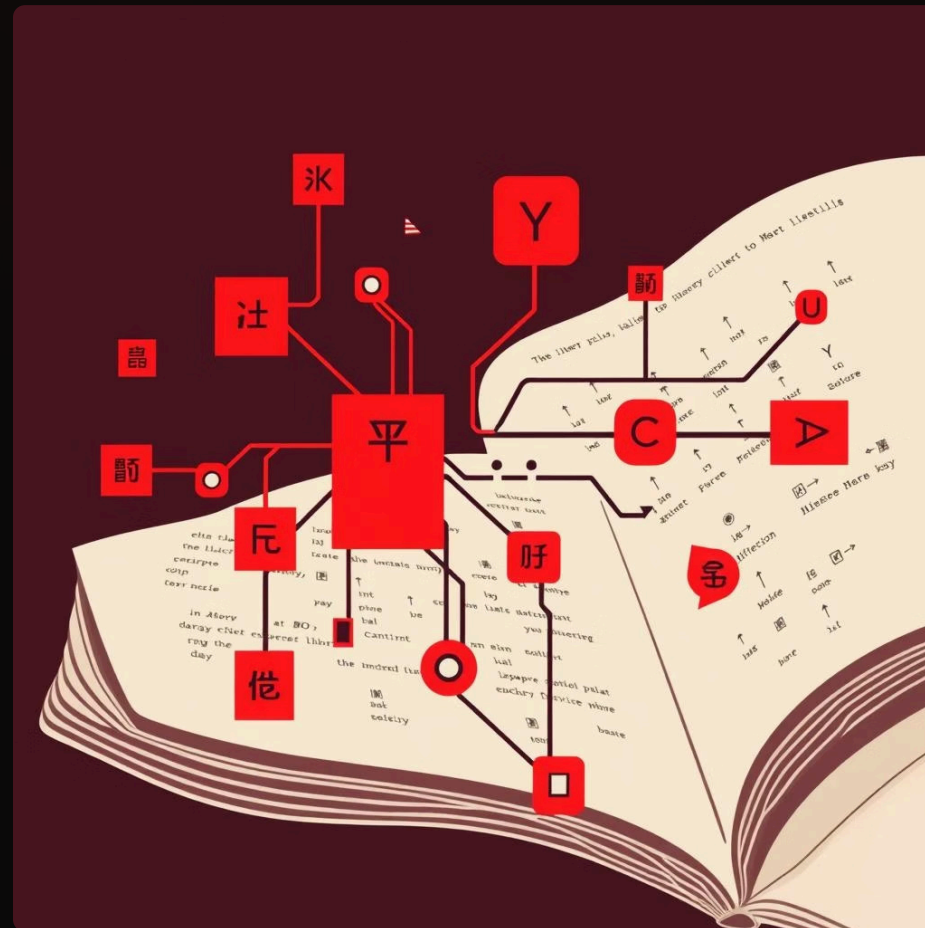
Eles são extremamente eficientes para buscar, adicionar ou remover dados quando você sabe a "chave" da informação que procura.

```
dicionario_vazio = {} # Cria um dicionário vazio
```

```
dicionario_vazio1 = dict() # Cria um dicionário vazio
```

```
aluno = {"nome": "Ana", "idade": 20, "curso": "Engenharia"}
```

Neste exemplo, **"nome"**, **"idade"** e **"curso"** são as **chaves**, e **"Ana"**, **20** e **"Engenharia"** são os respectivos **valores**.



Manipulação de Dicionários

Vamos ver como é fácil interagir com dicionários em Python. As operações são intuitivas e poderosas:



Criar e Adicionar

Você pode criar um dicionário vazio e ir adicionando novos pares, ou criá-lo já com dados. Adicionar um novo item é tão simples quanto atribuir um valor a uma nova chave.



Acessar e Atualizar

Acesse um valor usando sua chave entre colchetes. Se a chave já existir, o valor será atualizado; caso contrário, será adicionado.



Remover Itens

Utilize a palavra-chave `del` para remover um par chave-valor específico de um dicionário.

```
aluno["idade"] = 21 # Atualiza a idade
aluno["nota"] = 9.5 # Adiciona uma nova chave-valor
del aluno["curso"] # Remove o curso
```

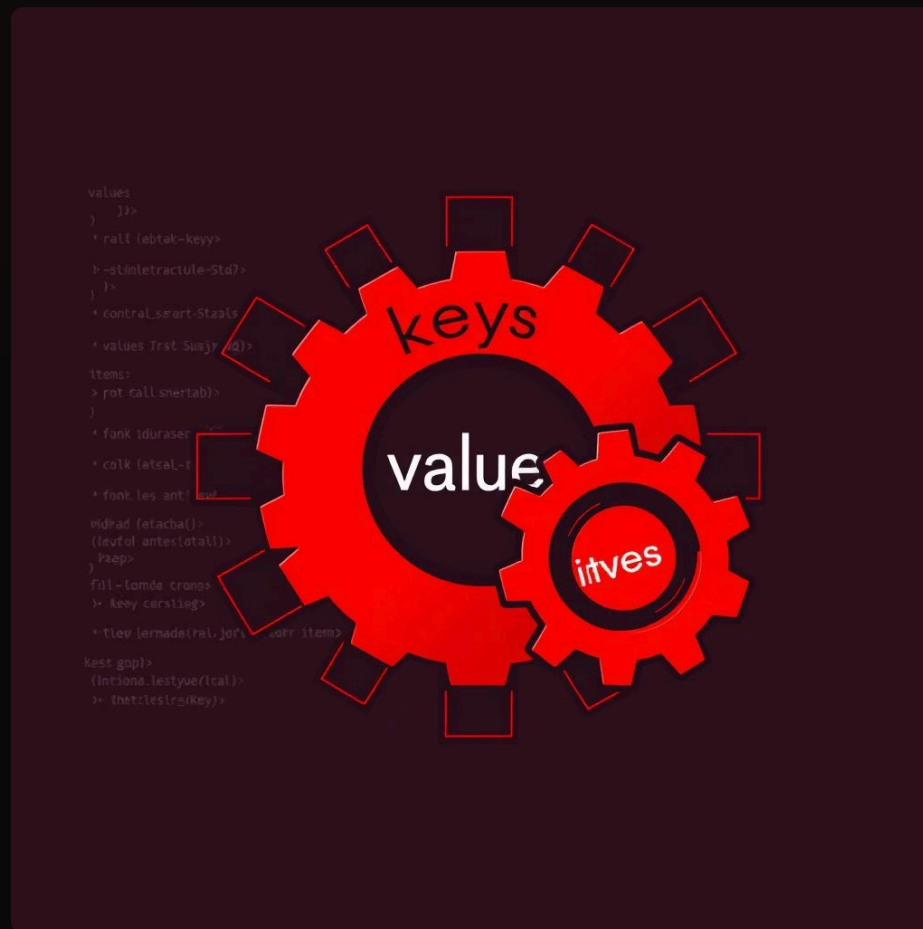
Métodos Úteis para Dicionários

Python oferece métodos práticos para explorar o conteúdo dos dicionários, facilitando a manipulação e a inspeção dos dados.

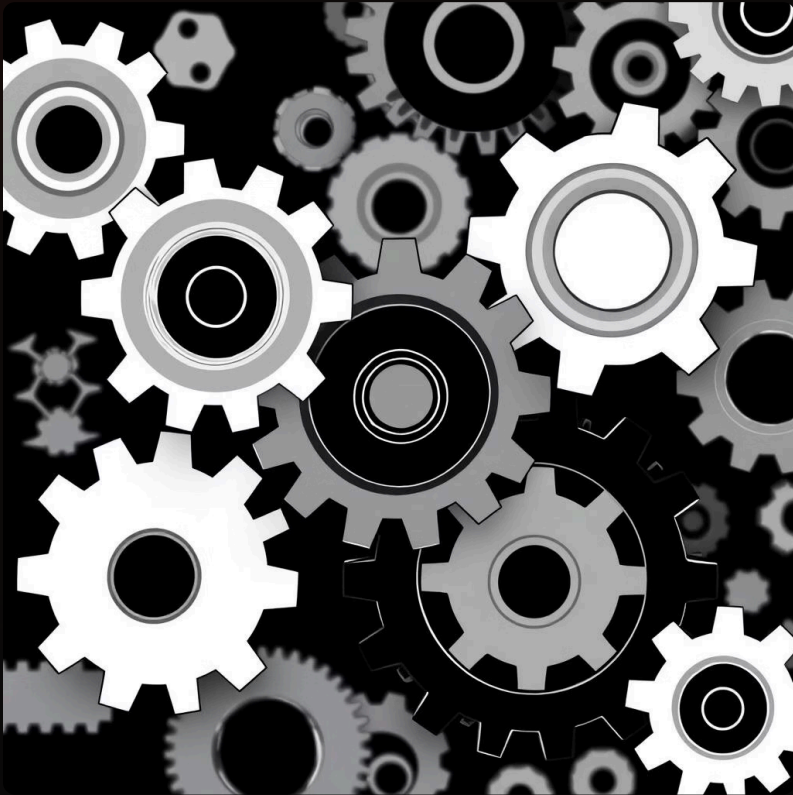
- **keys()**: Retorna uma "visão" de todas as **chaves** no dicionário. Perfeito para iterar sobre elas.
- **values()**: Retorna uma "visão" de todos os **valores**. Útil para processar apenas os dados sem as chaves.

```
print(aluno.keys()) # 'nome', 'idade', 'nota'
```

```
print(aluno.values()) # 'Ana', 21, 9.5
```



Métodos Úteis para Dicionários



Percorrendo um dicionário:

Para percorrer um dicionário, utilizamos o **laço de repetição for**, semelhante a forma com que fazemos com as listas. Porém, precisamos declarar **duas variáveis dentro do for** e utilizar o método **.items()** para conseguirmos acessar tanto as chaves quanto os valores do dicionário

- **items()**: Retorna uma "visão" de todos os pares (**chave**, **valor**). Ideal para iterar sobre ambos.

```
for chave, valor in dicionario.items():  
    print(chave, valor)
```


Listas vs. Dicionários: Quando Usar Cada Um?

Embora ambos armazenem coleções de dados, suas estruturas e propósitos são bem diferentes.

Listas: Sequências Ordenadas

- Armazenam itens em uma **ordem específica**.
- Acessados por **índices numéricos** (0, 1, 2...).
- Permitem **elementos duplicados**.
- Ideal para coleções de **itens onde a ordem importa**, como uma lista de compras ou uma sequência de números.

```
minha_lista = [10, "Python", 3.14]
```

Dicionários: Mapeamentos Chave-Valor

- Armazenam itens como pares **chave: valor**.
- Acessados por **chaves descritivas** (strings, números, tuplas).
- Cada chave deve ser **única**.
- Ideal para representar objetos com propriedades nomeadas, como informações de um usuário ou configurações de um programa.


```
meu_dicionario = {"nome": "Maria", "idade": 25}
```


Desafio Final: Sistema de Login Simplificado

Vamos aplicar o que aprendemos para criar um sistema de login básico.

O que você precisa fazer:

1. Crie um dicionário para armazenar **usuários** como chaves e **senhas** como valores.
2. Solicite ao usuário que digite seu nome de usuário e senha.
3. Verifique se o usuário existe no dicionário e se a senha corresponde.
4. Exiba uma mensagem de "**Acesso Permitido!**" ou "**Acesso Negado!**".

 Pense em como você pode usar os métodos `keys()` ou simplesmente a verificação de existência de chave (`if chave in dicionario:`) para resolver este desafio!

