Fundamentos de Programação em Python

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis Mayr Lima Martins

Dicionários em Python

Vamos mergulhar em uma das estruturas de dados mais poderosas e versáteis do Python: os dicionários. Imagine precisar organizar informações complexas – nomes, idades, cidades, ou até mesmo detalhes sobre seus filmes favoritos. Os dicionários oferecem uma maneira eficiente e flexível de armazenar e acessar esses dados.

Objetivos de Aprendizagem

- Compreender o conceito de dicionários como estruturas de dados em pares chave-valor.
- Criar, modificar e acessar elementos em um dicionário.
- Aplicar dicionários para resolver problemas práticos comuns.

Conceito de Dicionários em Python

Um dicionário é uma coleção não ordenada de pares chave-valor. As chaves são únicas e imutáveis, enquanto os valores podem ser de qualquer tipo. Pense em um catálogo telefônico – você procura uma pessoa pelo nome (chave) para obter o número de telefone (valor).

```
+-----------------------------------------+
|     Agenda Telefônica (Dicionário)      |
+-----------------------------------------+
| "Ana"        -->   "(11) 91234-5678"    |
| "Carlos"     -->   "(21) 99876-4321"    |
| "Beatriz"    -->   "(31) 98765-1234"    |
+-----------------------------------------+
```

Vamos criar nosso primeiro dicionário em uma variável `meu_dicionario` no formato abaixo:

```
+------------------------------+
|        meu_dicionario        |
+------------------------------+
```

```
| "nome"       -->     "Ana"     |
| "idade"      -->     25        |
| "cidade"     -->  "Campinas"   |
+--------------------------------+
```

```
meu_dicionario = {
    "nome": "Ana",
    "idade": 25,
    "cidade": "Campinas"
}
```

Usamos a chave para acessar o valor correspondente:

```
meu_dicionario["nome"] # Saída: Ana
```

E podemos também usar o método `get()`:

```
meu_dicionario.get("nome") # Saída: Ana
```

Alternativamente, podemos utilizar um valor padrão no método `get()` em caso não haja um item no dicionário com a chave passada como parâmetro:

```
meu_dicionario.get("pais", "Não encontrado")  # Valor padrão
```

Isso ajuda a evitar o lançamento de um `KeyError` como no caso a seguir, onde a chave `estado` (não existente no dicionário) é passada:

```
meu_dicionario["estado"] # Erro
```

Operações Básicas

Assim como vimos na manipulação de strings e listas, a função `len()`retorna o número de pares chave-valor presentes no dicionário, fornecendo uma maneira rápida e eficiente de determinar seu tamanho ou a quantidade de dados que ele contém. Essa informação pode ser útil em diversas situações, como verificar se um dicionário está vazio ou controlar o limite do número de itens que um dicionário pode armazenar.

```
len(meu_dicionario) # Saída: 3
```

Após a inicialização, podemos modificar e/ou expandir o conteúdo de um dicionário, através da atribuição direta de novos valores às chaves existentes ou da introdução de novas chaves com seus respectivos valores.

Por exemplo, o código abaixo modifica o valor da chave `"idade"` (originalmente, 25) para 26:

```
meu_dicionario["idade"] = 26  # Modifica
meu_dicionario
```

Para adicionar um novo item (um novo par chave-valor), basta atribuir um valor à uma nova chave no dicionário:

```
meu_dicionario["profissao"] = "Médica" # Adiciona novo item
meu_dicionario
```

Removendo items: Python oferece diversas maneiras de realizar a remoção de itens em dicionários. Veremos 3 opções a seguir: `del`, `pop()` e `clear()`.

Usando a operação `del` permite remover um item do dicionário especificando sua chave. Essa operação modifica o dicionário diretamente, removendo a entrada correspondente:

```
del meu_dicionario["cidade"] # Remove o item referente à chave "cidade"
meu_dicionario
```

Note que executar a operação `del` em uma chave que já foi removida ou que não existe lança um `KeyError`:

```
del meu_dicionario["cidade"] # Lança um erro ao tentar remover item inexistente
meu_dicionario
```

Alternativamente, o método `pop()` remove a chave especificada e retorna o valor associado a ela. Essa abordagem é útil quando você precisa tanto remover a chave quanto obter o valor que estava armazenado:

```
profissao = meu_dicionario.pop("profissao")
profissao
```

Finalmente, para remover todos os itens do dicionário de forma imediata, utilizamos o método `clear()`:

```
meu_dicionario.clear()
meu_dicionario
```

Percorrendo um Dicionário em um Loop

A capacidade de iterar sobre os elementos de um dicionário é muito importante para processar e manipular seus dados. Python oferece diversas maneiras de percorrer um dicionário, permitindo que você acesse tanto as chaves quanto os valores, ou ambos.

Vamos criar um novo dicionário para percorrê-lo utilizando um loop `for`:

```
star_wars = {
    "titulo": "Star Wars: Uma Nova Esperança",
    "ano_lancamento": 1977,
    "diretor": "George Lucas",
    "atores": ["Mark Hamill", "Harrison Ford", "Carrie Fisher"],
    "genero": ["Aventura", "Ficção Científica", "Fantasia"],
    "sinopse": "Em uma galáxia muito, muito distante...",
    "avaliacao_imdb": 8.7,
    "musica_tema": "The Imperial March"
}
```

Percorrendo Chaves: O método `keys()` retorna uma "view" das chaves do dicionário, que pode ser iterada diretamente em um loop `for`. Isso permite acessar cada chave individualmente para realizar operações específicas.

```
for k in star_wars.keys():
    print(k)
```

Percorrendo Valores: De forma similar, o método `values()` retorna uma "view" dos valores do dicionário, permitindo a iteração e processamento de cada valor sem acesso à chave correspondente.

```
for v in star_wars.values():
    print(v)
```

Percorrendo Chave e Valor: Para acessar tanto a chave quanto o valor simultaneamente, o método `items()` é utilizado. Ele retorna uma "view" dos pares chave-valor como tuplas, que podem ser desempacotadas diretamente no loop `for`, facilitando a manipulação de ambos os elementos em conjunto. A utilização do f-string (f"{chave}: {valor}") permite formatar a saída para melhor legibilidade.

```
for k, v in star_wars.items():
    print(f"{k} -> {v}")
```

Dicionários Contendo Tipos de Dados Complexos

Até agora, exploramos dicionários onde os valores eram geralmente simples: strings ou números. No entanto, a verdadeira flexibilidade dos dicionários em Python reside na sua capacidade de armazenar valores de tipos de dados mais complexos. Nesta seção, vamos introduzir um dicionário que demonstra essa capacidade, contendo valores como listas e outros dicionários aninhados.

Você pode representar estruturas de dados muito mais sofisticadas dentro de um único dicionário. Por exemplo, podemos ter um dicionário que representa uma coleção de produtos, onde cada produto é representado por um dicionário interno com informações detalhadas (nome, preço, características) e a coleção inteira é armazenada em uma lista de dicionários.

```
produto_exemplo = {
    "id": "PROD001",
    "nome": "Smartphone XYZ",
    "preco": 999.99,
    "caracteristicas": ["Câmera de 12MP", "Tela AMOLED", "8GB RAM"],
    "detalhes": {
        "fabricante": "TechCorp",
        "modelo": "XYZ-2023",
        "garantia": "1 ano"
    }
}
```

Observe como o valor associado à chave `"caracteristicas"` é uma lista, e o valor associado à chave `"detalhes"` é outro dicionário. Isso ilustra a capacidade de um dicionário armazenar estruturas de dados hierárquicas e complexas, permitindo representar informações mais ricas e organizadas. A próxima etapa será explorar como acessar e manipular esses valores complexos dentro do dicionário.

Vamos realizar algumas operações neste dicionário mais complexo:

```
# Acessando valores:
print(f"ID do produto: {produto_exemplo['id']}")
print(f"Preço: {produto_exemplo['preco']}")

# Acessando elementos dentro de uma lista:
print(f"Primeira característica: {produto_exemplo['caracteristicas'][0]}")

# Acessando informações detalhadas sobre o fabricante:
print(f"Fabricante: {produto_exemplo['detalhes']['fabricante']}")
```

Note que podemos adicionar uma nova característica à lista de características apenas referenciando-a com usa chave correspondente:

```
# Adicionando uma nova característica à lista:
produto_exemplo['caracteristicas'].append("Bateria de Longa Duração")
produto_exemplo['caracteristicas']
```

De forma similar, podemos modificar a duração da garantia no dicionário aninhado de `detalhes`:

```
# Modificando a garantia:
produto_exemplo['detalhes']['garantia'] = "2 anos"
produto_exemplo['detalhes']
```

Pergunta: Como você faria para imprimir todas as informações do dicionário `produto_exemplo`, exceto o valor associado à chave `"preco"`?

Onde Usar Dicionários

Alguns exemplos de usos de dicionários em casos reais são:

- **Configurações de Aplicação:** Armazenar configurações (ex: nome do banco de dados e seu endereço de acesso).
- **Mapear dados**: RA para nome dos alunos, nomes de produtos para seus respectivos preços etc.
- **Trabalhar com APIs**: Manipular dados estruturados em formato JSON.

Exercícios para Praticar:

1. Contador de Palavras:

- **Objetivo:** Criar um programa que conte a frequência de cada palavra em uma string fornecida pelo usuário.
- **Instruções:**
  - Solicite ao usuário que insira uma frase ou parágrafo.
  - Converta a frase para minúsculas e remova pontuações (você pode usar expressões regulares para isso).
  - Use um dicionário para armazenar a contagem de cada palavra.
  - Itere sobre as palavras da frase e atualize o contador no dicionário.
  - Imprima o dicionário resultante, mostrando a frequência de cada palavra.

2. **Criador de Um Dicionário de Frequências:**

- **Objetivo:** Criar um programa que recebe uma lista de números como entrada e gera um dicionário onde as chaves são os números da lista e os valores são suas respectivas frequências.
- **Instruções:**
  - Solicite ao usuário que insira uma lista de números separados por espaços.
  - Converta a string em uma lista de inteiros ou floats.
  - Use um dicionário para armazenar as frequências dos números.
  - Itere sobre a lista e, para cada número, incremente sua contagem no dicionário.
  - Imprima o dicionário resultante.

3. **Dicionário de Produtos:**

- **Objetivo:** Simular um sistema simples de gerenciamento de produtos usando um dicionário.
- **Instruções:**
  - Crie um dicionário onde as chaves são os nomes dos produtos (strings) e os valores são dicionários contendo informações sobre cada produto, como preço, quantidade em estoque e categoria, conforme o exemplo abaixo:

```
produtos = {
    "Camiseta": {
        "preco": 29.99,
        "quantidade_em_estoque": 50,
        "categoria": "Vestuário"
    },
    "Calça Jeans": {
        "preco": 79.99,
        "quantidade_em_estoque": 30,
        "categoria": "Vestuário"
    },
    "Livro Python para Iniciantes": {
        "preco": 45.50,
        "quantidade_em_estoque": 20,
```

```
        "categoria": "Livros"
    },
}
```

- Implemente funções para:
  - Adicionar um novo produto ao dicionário.
  - Atualizar o preço ou a quantidade de um produto existente.
  - Remover um produto do dicionário.
  - Buscar informações sobre um produto específico.
  - Listar todos os produtos no dicionário.

Conclusão

O que aprendemos hoje?

- Dicionários são estruturas essenciais para associar informações de forma organizada.
- Permitem buscas rápidas e operações eficientes com dados nomeados.
- Dominar dicionários é fundamental para trabalhar com dados reais e APIs.

Próximos Passos

- Resolva os problemas na seção "Exercícios para Praticar".
- Entenda a relação entre dicionários em Python e Hash maps/tables