

Python: Coleções e estruturas de dados

Estruturas de dados





Estruturas de dados

- Python apresenta diversos objetos que permitem manipular estruturas de dados.
- Estruturas de dados tradicionais em programação são representados por tipos baseados em coleções.
- O uso destes tipos contribui para reduzir a necessidade de criar e manter estruturas de forma manual.



Estruturas de dados

- Os tipos podem ser divididos em:
 - Sequenciais: Tuplas, listas e arrays.
 - Conjuntos: Tipo específico Set.
 - Mapeamento: Dicionários
- Cada tipo possui um conjunto de propriedades e métodos.
- Alguns tipos permitem iterar sobre os dados e acessar por deslocamento.
- Existem estruturas mutáveis e imutáveis.



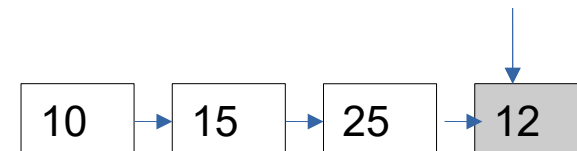
Listas (Tipo list)

- Sequencias ordenadas de zero ou mais referências para objetos, desta forma, cada elemento pode ser de tipo diferente.
- Listas são mutáveis, ou seja, é possível inserir, alterar, substituir, retirar elementos.
- Para criar uma lista é possível usar o método `list()` ou simplesmente usar colchetes em uma atribuição.

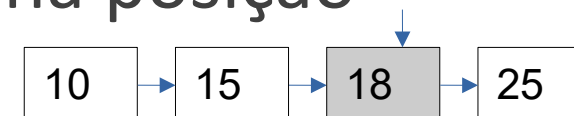
Listas (Tipo list)

- Alguns métodos:

- `append(x)`: Inclui no fim da lista



- `insert(i, x)`: Insere um elemento na posição determinada por `i`



- `count(x)`: Quantas vezes um elemento aparece na lista

- `pop()` e `pop(i)`: Retorna e remove elemento no fim ou na posição atual

- `Sort(..)`: Ordena uma lista



Listas (Tipo list)

- Fatiamiento permite acessar partes de listas usando coordenadas.
- `lista[inicio:fim:intervalo]`
- Quando não especificados, início e fim consideram a posição 0 e a última posição, respectivamente.
- Intervalo padrão é 1.
- É possível usar valores negativos para obter em ordem inversa.
- É possível ainda usar operador `*` para desempacotar listas e outras estruturas



List comprehension

- Geração de listas usando expressões.
- `[var for var in collection if..]`
- Permite utilização de instruções condicionais para selecionar valores que estarão na lista gerada.
- Recurso muito utilizado por programadores mais experientes.
- Comprehension é aplicável a diferentes estruturas de dados.



Tuplas

- Semelhantes à listas, porém imutáveis.
- São sequencias ordenadas de zero ou mais referências para objetos.
- Podem ser criadas com a função `tuple()` ou com atribuições de múltiplos elementos para uma variável.
- A atribuição pode usar parênteses ou não.
- Suporta o fatiamento e iteração.
- Pacote `collections` possui recurso para criação de tuplas nomeadas.



Conjuntos

- Conjuntos são coleções desordenadas, mutáveis com referências para objetos que são hashtables (podem ser usados como chaves).
- Contém itens únicos, sem valores repetidos.
- Suportam iteração, associações e outras operações como união, intersecção, etc.
- Não podem ser acessados por deslocamento, ou seja, não podem ser fatiados.
- Criados com `set()` ou atribuição com chaves `{..}`



Conjuntos

- Alguns métodos:
 - `s.add(x)`: Adiciona elemento, se não existir.
 - `s.difference(x)`: Valores que estão em `s` e não em `x`.
 - `s.intersection(x)`: Valores que estão em `s` e `x`.
 - `s.union(x)`: União dos valores de `s` e `x`
 - `s.remove(x)`: Remove o valor `x` do conjunto `s`

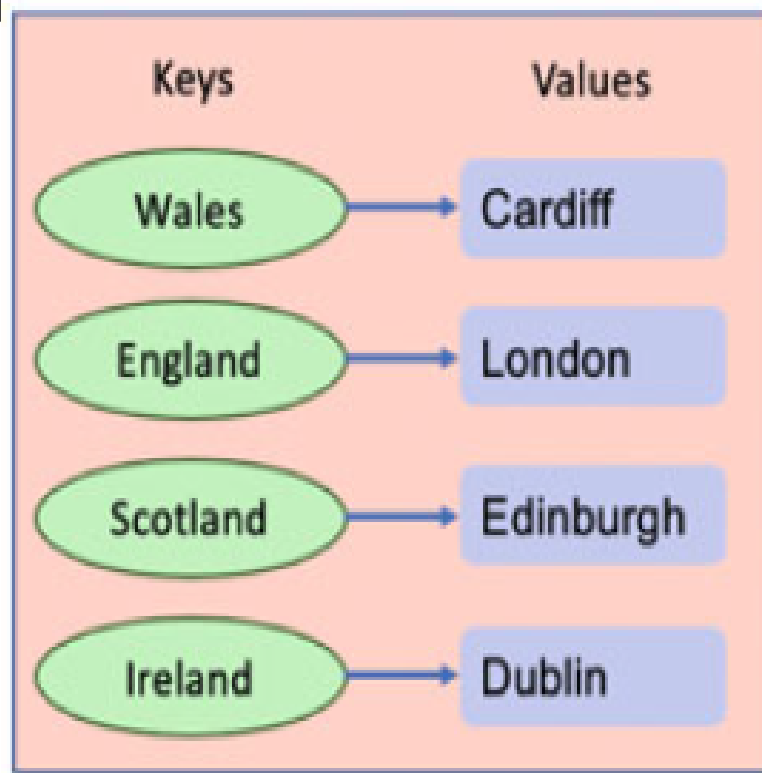


Dicionários

- Um dicionário é considerado um tipo baseado em mapeamento.
- Mapeamentos consistem em pares chave-valor.
- Exemplo: {'nome': 'Evandro'}, onde nome é a chave e Evandro o valor associado.
- Chaves devem possuir característica de ser hashtable.
- Tipos imutáveis, como int, float, string podem ser usados como chave, porém, list, dict, set que são mutáveis não são hashtable.

Dicionários

- A cada chave está associado um objeto, porém, este pode ser uma lista, conjunto ou outro objeto com múltiplas referências.





Dicionários

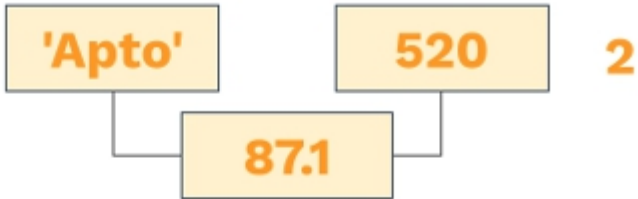

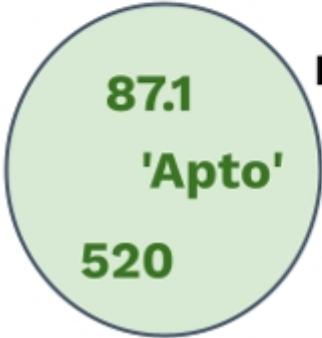
- Dicionário pode ser criado por `dict()` ou usando `{}` na atribuição.
- Cada chave em um dicionário é única, por isso um atribuição com uma chave já existente envolve a substituição do valor anterior.
- Assim como em listas, é possível usar expressões para dict comprehensions.
- A iteração pode ser pelas chaves (`keys()`), valores (`values()`) ou pelo item completo (`items()`).
- Não permitem fatiamento, nem o acesso por posição.



Dicionários

- Algumas funções:
 - `get(x)`: Retorna a chave para o valor `x`.
 - `items()`: Lista dos pares chave-valor.
 - `keys()`: Lista das chaves.
 - `values()`: Lista dos valores.
 - `pop(k)`: Retorna e remove o item da chave `k`

Resumindo

<p>índice: 0</p> <p>Lista</p>  <p>Ordenável e Mutável 1 Exemplos: <code>list(1, 2)</code> <code>['Apto', 87.1, 520]</code></p>	<p>Tupla</p>  <p>índice: 0 1 2</p> <p>Ordenável e Imutável Exemplos: <code>tuple(1, 2)</code> <code>(1, 2)</code> 1, 2</p>								
<p>Dicionário</p> <p>Não ordenável Chave imutável Exemplos: <code>dict({1 : 2})</code> <code>{ '2021' : 25, '2020' : 100, '2019' : 94 }</code></p> <table><thead><tr><th>chave</th><th>valor</th></tr></thead><tbody><tr><td>'2021'</td><td>25</td></tr><tr><td>'2020'</td><td>100</td></tr><tr><td>'2019'</td><td>94</td></tr></tbody></table>	chave	valor	'2021'	25	'2020'	100	'2019'	94	<p>Conjunto</p>  <p>Não ordenável e Mutável Exemplos: <code>set({1, 2})</code> <code>{ 'Apto', 87.1, 520 }</code></p>
chave	valor								
'2021'	25								
'2020'	100								
'2019'	94								



Pacote collection

- O pacote collection apresenta diversos recursos que ampliam as estruturas de dados nativas do Python.
- Em sua maioria as classes estendem e combinam funcionalidades das estruturas básicas.
- O acesso a este pacote se dá pela instrução `import collections` ou `from collections import ...`



Pacote collection

- Recursos principais:
 - Namedtuple: Cria tuplas nomeadas, que funcionam como registros.
 - Defaultdict: Derivada de dict, trata valores múltiplos e valores faltantes em dicionários.
 - Deque: Listas que permitem inserção em ambos os lados, diferentemente de listas encadeadas tradicionais.
 - Orderdict: Dicionários ordenados.
 - Counter: Dicionário com chaves e contagens de incidência por chave.