Princípios de Construção de Algoritmos

AULA 07 - Collections em Python

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/fundamentos-em-tic

Agenda

1. Introdução às Collections;
2. Listas (Lists);
3. Tuplas (Tuples);
4. Conjuntos (Sets);
5. Dicionários (Dictionaries);
6. Collections Especializadas:
    - Counter;
    - defaultdict;
    - OrderedDict;
    - namedtuple;
    - deque;
7. Arrays;
8. Comparação entre Collections;
9. Exercícios Práticos.

Introdução às Collections

- Estruturas de dados para armazenar múltiplos itens;
- Fundamentais para organização e manipulação eficiente de dados;
- Tipos principais: **listas, tuplas, conjuntos, dicionários**;
- Collections especializadas do módulo `collections;`
- Importância na programação Python.

Listas (*Lists*)

- Sequências mutáveis e ordenadas;
- Criação: `my_list = [1, 2, 3];`
- Acesso: `my_list[0];`
- Slicing: `my_list[1:3].`

Listas (continuação)

Métodos principais:

- `append(x)`: Adiciona um item ao final;
- `extend(iterable)`: Adiciona todos os itens do iterável;
- `insert(i, x)`: Insere um item em uma posição específica;
- `remove(x)`: Remove a primeira ocorrência de um item;
- `pop([i])`: Remove e retorna o item em uma posição (ou o último);
- `sort()`: Ordena a lista in-place;
- `reverse()`: Inverte a ordem dos elementos.

Listas (continuação)

List comprehensions:

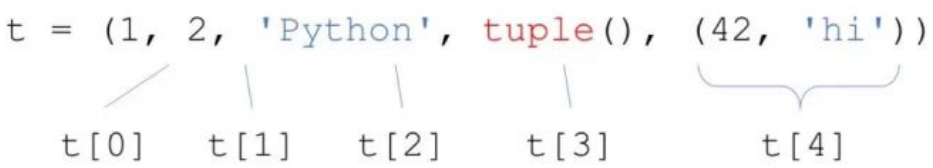
```
squares = [x**2 for x in range(10)]
even_squares = [x**2 for x in range(10) if x % 2 == 0]
```

- Vantagens:
  - Versatilidade;
  - Fácil de modificar;
  - Suporta indexação e *slicing*;

- Desvantagens:
  - Uso de memória pode ser ineficiente para grandes conjuntos de dados.

Tuplas (*Tuples*)

- Sequências imutáveis;
- Criação: `my_tuple = (1, 2, 3)` ou `my_tuple = 1, 2, 3;`
- Acesso: `my_tuple[0]`.

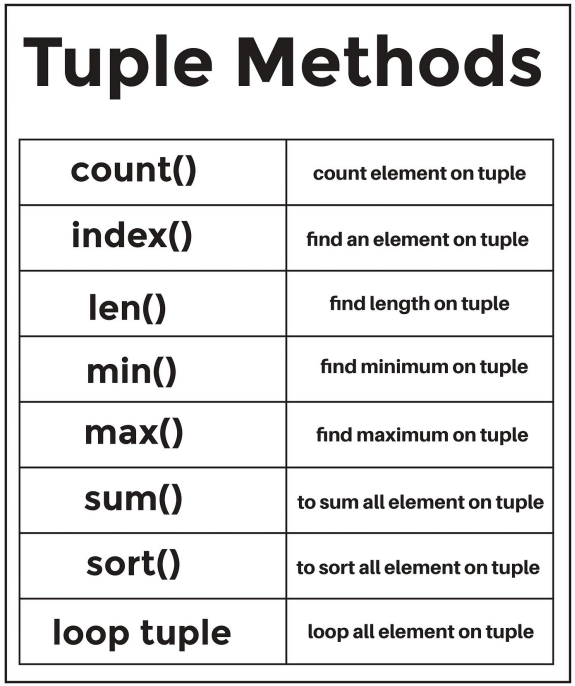


Tuplas (continuação)

- Desempacotamento:

```
x, y, z = my_tuple
```

- Usos comuns:
  - Retorno de múltiplos valores de funções;
  - Dicionários (como chaves);
  - Dados que não devem ser alterados.
- Vantagens:
  - Imutabilidade (segurança e *hashable*);
  - Mais eficientes em memória que listas;
- Desvantagens:
  - Não podem ser modificadas após a criação.



Conjuntos (*Sets*)

- Coleções não ordenadas de elementos únicos;
- Criação: `my_set = {1, 2, 3}` ou `set([1, 2, 3];`
- Não suporta indexação ou *slicing*;
- Métodos principais:
  - `add(x)`: Adiciona um elemento
  - `remove(x)`: Remove um elemento (erro se não existir)
  - `discard(x)`: Remove um elemento (sem erro se não existir)
  - `pop()`: Remove e retorna um elemento arbitrário

Conjuntos (continuação)

- Operações de conjunto:

- `union(other_set)`: União de conjuntos
- `intersection(other_set)`: Interseção de conjuntos
- `difference(other_set)`: Diferença entre conjuntos
- `symmetric_difference(other_set)`: Elementos em um ou outro, mas não em ambos.

- Vantagens:

- Remoção automática de duplicatas
- Operações de conjunto eficientes - Teste de pertencimento rápido

- Desvantagens:

- Não mantém ordem
- Não suporta indexação

Dicionários (Dictionaries)

- Estruturas de pares chave-valor
- Criação: `my_dict = {'chave': 'valor'}`
- Acesso: `my_dict['chave']` ou `my_dict.get('chave')`
- Métodos principais:

- `keys()`: Retorna uma vista das chaves
- `values()`: Retorna uma vista dos valores
- `items()`: Retorna uma vista dos pares chave-valor
- `update(other_dict)`: Atualiza o dicionário com outro
- `pop(key)`: Remove e retorna o valor de uma chave

Dicionários (continuação)

Dictionary comprehensions:

```
squares = {x: x**2 for x in range(5)}
```

Vantagens:

- Acesso rápido por chave
- Flexibilidade nas chaves e valores - Representação natural de estruturas de dados

Desvantagens:

- Uso de memória pode ser alto;
- Não mantém ordem (antes do Python 3.7).

Collections Especializadas

Counter

- Subclasse de dict para contagem de *hashables*;
- Criação: `Counter(['a', 'b', 'c', 'a', 'b', 'b'])`

Métodos principais:

- `most_common([n])`: Retorna os n elementos mais comuns
- `update([iterable-or-mapping])`: Atualiza contagens
- `subtract([iterable-or-mapping])`: Subtrai contagens

Collections Especializadas (continuação)

defaultdict

- Subclasse de dict que chama uma factory function para chaves ausentes
- Criação: `defaultdict(list)` ou `defaultdict(int)`

Exemplo:

```
from collections import defaultdict

d = defaultdict(list)
d['key'].append(1)  # Não levanta KeyError
```

Vantagens:

- Evita verificações de existência de chaves
- Útil para agrupar dados

Collections Especializadas (continuação)

OrderedDict

- Subclasse de dict que lembra a ordem de inserção das chaves
- Criação: `OrderedDict([('a', 1), ('b', 2), ('c', 3)])`

Métodos específicos:

- `move_to_end(key, last=True)`: Move uma chave para o início ou fim
- `popitem(last=True)`: Remove e retorna um par (chave, valor) do início ou fim.

Nota: Dicionários padrão preservam a ordem desde Python 3.7, mas OrderedDict ainda tem usos específicos.

Collections Especializadas (continuação)

namedtuple

- Função *factory* para criar subclasses de tupla com campos nomeados;
- Criação: `Point = namedtuple('Point', ['x', 'y'])`

Exemplo:

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p.x, p.y)  # 11 22
```

Vantagens:

- Código mais legível;

- Compatível com tuplas regulares.

6. Collections Especializadas (continuação)

deque

- Lista otimizada para inserções e deleções em ambas as extremidades
- Criação: `deque(['a', 'b', 'c'])`

Métodos principais:

- `append(x)`: Adiciona à direita
- `appendleft(x)`: Adiciona à esquerda
- `pop()`: Remove e retorna da direita
- `popleft()`: Remove e retorna da esquerda
- `rotate(n)`: Rotaciona os elementos

Vantagens:

- Eficiente para filas e pilhas;
- *Thread-safe*.

Arrays

- Semelhantes a listas, mas para tipos específicos;
- Módulo `array`: `from array import array;`
- Criação: `arr = array('i', [1, 2, 3]);`

Características:

- Mais eficientes em memória que listas para grandes quantidades de dados numéricos;

- Suportam muitas operações de lista;

- Tipagem restrita (todos os elementos devem ser do mesmo tipo).

Arrays (continuação)

Tipos comuns:

- ‘i’: inteiros signed
- ‘I’: inteiros unsigned
- ‘f’: float
- ‘d’: double

Exemplo:

```
from array import array

numbers = array('i', [1, 2, 3, 4, 5])
numbers.append(6)
numbers.extend([7, 8, 9])
```

Vantagens:

- Eficiência de memória
- Bom para operações numéricas intensivas

Desvantagens:

- Menos flexíveis que listas (tipo único)

8. Comparação entre Collections

Collection	Ordenada	Mutável	Acesso	Duplicatas
Lista	Sim	Sim	Índice	Sim
Tupla	Sim	Não	Índice	Sim
Conjunto	Não	Sim	Valor	Não
Dicionário	Sim*	Sim	Chave	Não (chaves)

*Ordenado a partir do Python 3.7.

Comparação entre Collections (continuação)

Escolha baseada em:

- Necessidade de modificação (mutável vs. imutável);
- Importância da ordem;
- Tipo de acesso necessário (índice, chave, valor);
- Presença de duplicatas;
- Eficiência de operações específicas;
- Uso de memória.

A *collection* certa pode melhorar significativamente o desempenho e a clareza do código.

Exercícios Práticos

Listas

1. Implemente uma função que receba uma lista de números e retorne uma nova lista com apenas os números pares, em ordem reversa.

Tuplas

2. Crie uma função que receba uma lista de tuplas representando coordenadas (x, y) e retorne a tupla com a maior distância da origem (0, 0).

Exercícios Práticos (continuação)

Conjuntos

3. Implemente um verificador de anagramas que use conjuntos para determinar se duas palavras são anagramas uma da outra, ignorando espaços e pontuação.

Dicionários

4. Crie um programa que simule um carrinho de compras, usando um dicionário para armazenar os itens (nome do produto como chave e quantidade como valor). Implemente funções para adicionar itens, remover itens e calcular o total da compra.

Exercícios Práticos (continuação)

Collections Especializadas

5. Use a classe Counter para analisar a frequência de palavras em um texto longo e encontrar as N palavras mais comuns, ignorando palavras comuns como “o”, “a”, “de”.
6. Implemente um cache simples usando OrderedDict que armazene os N itens mais recentemente acessados, removendo o item mais antigo quando o cache estiver cheio.

Exercícios Práticos (continuação)

Arrays

7. Implemente uma função que use arrays para calcular a média móvel de uma série de números. A função deve aceitar a série como uma lista e o tamanho da janela para a média móvel.

Considerações Finais

- As collections em Python oferecem ferramentas poderosas para manipulação de dados;
- A escolha da collection certa depende do caso de uso específico;
- Compreender as características de cada collection é crucial para escrever código eficiente;
- Prática e experimentação são essenciais para dominar o uso de collections.

Recursos Adicionais

- Documentação oficial do Python: https://docs.python.org/3/tutorial/datastructures.html
- PEP 3119 - Introducing Abstract Base Classes: https://www.python.org/dev/peps/pep-3119/
- Livro “Fluent Python” de Luciano Ramalho
- Python Collections Module: https://docs.python.org/3/library/collections.html

Princípios de Construção de Algoritmos

AULA 07 - Collections em Python

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/fundamentos-em-tic