

Departamento de Engenharia de Sistemas e Computação

Dicionários e Tuplas

Dados associativos e sequências imutáveis

João Araujo

Algoritmos Computacionais I

Aula 20

Roteiro

- O problema de buscar por *nome*, não por posição
- **Dicionários**: pares chave–valor (Cap. 5.12)
- **Tuplas**: sequências imutáveis (5.13)
- Tuplas como chaves; listas de tuplas; dicionários de dicionários

Listas ou dicionários?

Lista

Acesso por **posição** (índice inteiro): `notas[2]`.
Boa para sequências de itens do mesmo tipo.

Dicionário

Acesso por **chave** (nome, código...): `turma['João']`.
Boa para associar informações.

Exemplos do dia a dia: agenda de telefones, tabela de preços, matrícula → nome, estado → sigla, elemento químico → massa atômica.

Criando e consultando dicionários

```
turma = {'Linus': 123456, 'João': 102344,
         'Dennis': 654321, 'Ada': 321456}
print(turma['João'])         # 102344
print(turma)                 # {'Linus': 123456, 'João': 102344, ...}
turma['Grace'] = 777777      # acrescenta / altera
print(len(turma))            # 5
```

- Sintaxe: `{chave: valor, ...}`; consulta com colchetes
- Atribuir a uma chave *nova* cria o par; a uma chave *existente* altera o valor
- Chave inexistente na consulta ⇒ `KeyError`

Fonte: `code/Cap5/prog.5.32_testadic.py`

Construindo um dicionário aos poucos

```
filmes = {}                                    # dicionário vazio
filmes[1975] = 'Em Busca do Cálice Sagrado'
filmes[1979] = 'A Vida de Brian'
filmes[1983] = 'O Sentido da Vida'
print(filmes)
```

```
{1975: 'Em Busca do Cálice Sagrado', 1979: 'A Vida de Brian', 1983: 'O Sentido da Vida'}
```

- Chaves podem ser números, strings ou tuplas (qualquer tipo *imutável*)
- Valores podem ser *qualquer* coisa: números, listas, outros dicionários

Fonte: `code/Cap5/prog.5.33_filmesmp.py`

Operações úteis

```
d = {'amazonas': 'AM', 'rio de janeiro': 'RJ'}
print('amazonas' in d)              # True (testa as CHAVES)
print(d.get('acre', '?'))           # '?'  (sem KeyError)
for chave in d:                     # percorre as chaves
    print(chave, '->', d[chave])
for chave, valor in d.items():      # chave e valor juntos
    print(chave, valor)
print(list(d.keys()), list(d.values()))
del d['amazonas']
```

- Exercícios 5.29 e 5.30: dicionário de estados → siglas; versão *case-insensitive* (use `.lower()`)

Fonte: `code/Cap5/prog.5.d_testadic1.py`

Padrão: contar ocorrências

```
texto = "abracadabra"
contagem = {}
for letra in texto:
    if letra in contagem:
        contagem[letra] += 1
    else:
        contagem[letra] = 1
print(contagem)     # {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}

# versão curta
contagem = {}
for letra in texto:
    contagem[letra] = contagem.get(letra, 0) + 1
```

Frequências de palavras, defeitos por máquina, votos por candidato... Tudo isso é este padrão.

Tuplas: sequências imutáveis

```
planetas = ('Mercúrio', 'Vênus', 'Terra', 'Marte',
            'Saturno', 'Júpiter', 'Urano', 'Netuno')
print(planetas[2:5])        # ('Terra', 'Marte', 'Saturno')
print(planetas[:3])
print(len(planetas))
planetas[0] = 'Sol'         # TypeError: não suporta atribuição
```

- Sintaxe com parênteses; indexação e fatiamento iguais aos das listas
- Imutável: não há append, remove, atribuição por índice
- Tupla de um elemento: `(5,)` (a vírgula é obrigatória)

Fonte: code/Cap5/prog.5.e_ipbytuplas.py

Para que servem tuplas?

- Agrupar valores relacionados que **não devem mudar**: coordenadas (x, y), data (d, m, a), cor (r, g, b)
- **Retorno múltiplo** de funções (Aula 11): `return x, y` devolve uma tupla
- Podem ser **chaves de dicionário** (listas não podem, pois são mutáveis)
- Mais leves e seguras: protegem contra alterações acidentais

Desempacotamento

```
x, y = (3, 4)          a, b = b, a  (troca de valores)
nome, *resto = ('Ana', 10, 8.5)     (resto é uma lista)
```

Combinando: dicionário de tuplas

```
lat_galeao = (22, 48, 34);  long_galeao = (43, 15, 0)
aeroporto_galeao = ('Galeão', lat_galeao, long_galeao)
lat_garulhos = (23, 26, 6); long_garulhos = (46, 28, 22)
aeroporto_garulhos = ('Guarulhos', lat_garulhos, long_garulhos)
aeroportos = {'Rio': aeroporto_galeao,
              'São Paulo': aeroporto_garulhos}
for cidade in aeroportos:
    nome, lat, lon = aeroportos[cidade]
    print(f"Aeroporto {cidade}: {nome}")
    print(f"  Latitude: {lat[0]}º{lat[1]}'{lat[2]}\"")
```

Estruturas aninhadas permitem modelar dados reais. Exercício 5.33 (aeroportos com tuplas).

Fonte: `code/Cap5/prog.5.37_aeroportos.py`

Registros: lista de dicionários

```
pecas = [
    {'id': 1000, 'maquina': 'M1', 'tempo': 31.6, 'status': 'Aprovado'},
    {'id': 1001, 'maquina': 'M1', 'tempo': 34.5, 'status': 'Aprovado'},
    {'id': 1002, 'maquina': 'M2', 'tempo': 24.9, 'status': 'Descarte'},
]
descartadas = [p for p in pecas if p['status'] == 'Descarte']
tempo_medio = sum(p['tempo'] for p in pecas) / len(pecas)

por_maquina = {}
for p in pecas:
    por_maquina.setdefault(p['maquina'], []).append(p['tempo'])
```

Uma lista de dicionários é uma "tabela com colunas nomeadas": a ideia que o **Pandas** formaliza (Aulas 28 e 29).

Resumo comparativo

	Lista	Tupla	Dicionário
Sintaxe	`[1, 2]`	`(1, 2)`	`{'a': 1}`
Acesso	índice	índice	chave
Mutável?	sim	não	sim
Ordenada?	sim	sim	mantém ordem de inserção
Pode ser chave?	não	sim	não
Uso típico	coleção de itens	registro fixo	associação chave–valor

Para praticar: exercícios do livro

Site do livro – Capítulo 5

- Dicionários: **5.29** (estados e siglas), **5.30** (sem diferenciar maiúsculas)
- Tuplas: **5.31** (tupla de divisíveis por 7), **5.32** (π com lista de tuplas), **5.33** (aeroportos)

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap5/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Dicionário: coleção de pares chave–valor; acesso rápido por chave
- Tupla: sequência imutável; ótima para registros fixos e retornos múltiplos
- Estruturas se combinam: listas de dicionários, dicionários de tuplas...
- Próxima aula: NumPy