Linguagem de Programação Python

Dicionários

Professor: Ritomar Torquato

1

Dicionários

Objetivos: Conhecer os fundamentos do tipo de dados Dicionários;

2

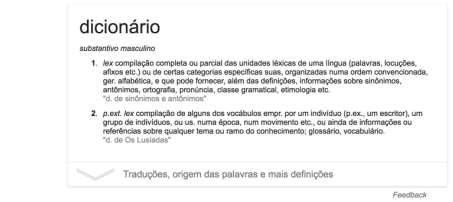
Introdução

Dicionários são estruturas de dados similares às listas, mas com propriedades de **acesso** diferentes.

3

Introdução

Dicionários são estruturas de dados similares às listas, mas com propriedades de **acesso** diferentes.



dicionário

substantivo masculino

1. *lex* compilação completa ou parcial das unidades léxicas de uma língua (palavras, locuções, afixos etc.) ou de certas categorias específicas suas, organizadas numa ordem convencionada, ger. alfabética, e que pode fornecer, além das definições, informações sobre sinônimos, antônimos, ortografia, pronúncia, classe gramatical, etimologia etc.
   "d. de sinônimos e antônimos"
2. *p.ext. lex* compilação de alguns dos vocábulos empr. por um indivíduo (p.ex., um escritor), um grupo de indivíduos, ou us. numa época, num movimento etc., ou ainda de informações ou referências sobre qualquer tema ou ramo do conhecimento; glossário, vocabulário.
   "d. de Os Lusíadas"

Traduções, origem das palavras e mais definições

Feedback

Relacionam chave e valor.

4

Introdução

Nas listas cada valor é acessado por seu índice.

[	Alface	Batata	Tomate	Feijão	]
	0	1	2	3	

[	0,45	1,20	2,30	1,50	]
	0	1	2	3	

Nos dicionários o valor é acessado por sua chave.

{	Alface	Batata	Tomate	Feijão	}
	0,45	1,20	2,30	1,50	

5

Dicionários

Python são **{ }**:

Produto	Preço
Alface	R$ 0,45
Batata	R$ 1,20
Tomate	R$ 2,30
Feijão	R$ 1,50

```
>>> tabela = {
        "Alface" : 0.45,
        "Batata" : 1.20,
        "Tomate" : 2.30,
        "Feijão" : 1.50,
    }
```

Cada elemento é uma combinação de chave e valor.

O valor é acessado por sua chave:

```
>>> tabela["Tomate"]
2.3
>>> print(f'Preço do feijão: {tabela["Feijão"]:.2f}')
Preço do feijão: 1.50
```

6

Dicionários

Quando atribuímos um valor à uma chave:

Caso 1) A chave não existe

```
>>> tabela = { "Alface": 0.45,
               "Batata": 1.20,
               "Feijão": 1.50 }
>>> tabela
{'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> tabela["Tomate"] = 2.50
>>> print(tabela)
{'Tomate': 2.5, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> print(tabela["Tomate"])
2.5
```

Uma nova chave é incluída e o valor é atribuído.

7

Dicionários

Quando atribuímos um valor à uma chave:

Caso 2) A chave já existe

```
>>> tabela = { "Alface": 0.45,
               "Batata": 1.20,
               "Tomate": 2.30,
               "Feijão": 1.50 }
>>> tabela
{'Tomate': 2.3, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> tabela["Tomate"] = 2.50
>>> print(tabela)
{'Tomate': 2.5, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> print(tabela["Tomate"])
2.5
```

O valor da chave é alterado.

8

Dicionários

Uma **KeyError** é lançada ao tentar acessar uma chave inexistente:

```
>>> tabela
{'Tomate': 2.5, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> tabela["Cebola"]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    tabela["Cebola"]
KeyError: 'Cebola'
>>> tabela["tomate"]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    tabela["tomate"]
KeyError: 'tomate'
>>> tabela["Tomate"]
2.5
```

9

Dicionários

Busca de valores segura com o método **.get()**:

```
>>> tabela
{'Tomate': 2.5, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> tabela.get("Cebola", "Produto não encontrado.")
'Produto não encontrado.'
>>> print(tabela.get("tomate", "Produto não encontrado."))
Produto não encontrado.
>>> tabela.get("Tomate", "Produto não encontrado.")
2.5
>>> print(tabela.get("tomate"))
None
```

O método **.get()** permite informar um valor padrão (*default*) que será retornado caso o valor buscado não seja encontrado e retorna **None** se não informar o valor padrão.

10

Dicionários

Verificação da existência de uma chave com **in**:

```
>>> "Manga" in tabela
False
>>> if "Batata" in tabela:
        print("Sopa de batatas")


Sopa de batatas
>>> tabela
{'Tomate': 2.5, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
```

11

Dicionários

Verificação da existência de uma chave com operador **in**:

```
>>> for item in tabela:
        print(f'{item}: R$ {tabela[item]:.2f}')


Alface: R$ 0.45
Batata: R$ 1.20
Tomate: R$ 2.30
Feijão: R$ 1.50
>>> print("Feijão" in tabela)
True
```

12

Dicionários

Obtenção de uma lista de chaves e valores.

```
>>> tabela = { "Alface": 0.45,
               "Batata": 1.20,
               "Tomate": 2.30,
               "Feijão": 1.50 }
>>> tabela.keys()
dict_keys(['Tomate', 'Batata', 'Alface', 'Feijão'])
>>> tabela.values()
dict_values([2.3, 1.2, 0.45, 1.5])
```

Os métodos **.keys()** e **.values()** retornam geradores().

Geradores são funções que se comportam como objetos iteráveis e podem ser utilizados diretamente em loops **for**.

13

Dicionários

Obtenção de uma lista de chaves e valores.

```
>>> tabela.keys()
dict_keys(['Tomate', 'Batata', 'Alface', 'Feijão'])
>>>
>>> tabela.values()
dict_values([2.3, 1.2, 0.45, 1.5])
>>>
>>> chaves = list(tabela.keys())
>>> valores = list(tabela.values())
>>> chaves
['Tomate', 'Batata', 'Alface', 'Feijão']
>>> valores
[2.3, 1.2, 0.45, 1.5]
```

Podem ser transformados em listas ou tuplas usando, respectivamente, **list()** e **tuple()**

14

Dicionários

É possível usar qualquer tipo para chaves ou valores.

```
alunos_por_media = {
        'Abaixo da média': ['Ana', 'Pedro'],
        'Na média': ['Bia', 'Maria', 'Clara'],
        'Acima da média': ['Dani', 'Alan', 'Borges'],
    }

notas_alunos = {
        1001 : ('Ana', 6.0, 4.0),
        1002 : ('Borges', 7.0, 8.0),
        1003 : ('Maria', 6.5, 7.0),
        1004 : ('Clara', 7.0, 5.0),
        1005 : ('Bia', 6.0, 6.0),
        1006 : ('Dani', 9.0, 10.0),
        1007 : ('Alan', 8.0, 9.0),
        1008 : ('Pedro', 5.5, 5.5),
    }
```

15

Dicionários

Obtendo itens.

```
>>> tabela.items()
dict_items([('Batata', (500, 1.2)), ('Tomate', (700, 2.3)), ('Feijão', (750, 1.5))])
>>> for chave in tabela:
        estoque, preco = tabela[chave]
        print(f'Produto: {produto}:\nEstoque: {estoque}:\nPreço: {preco}\n')

Produto: Feijão:
Estoque: 500:
Preço: 1.2

Produto: Feijão:
Estoque: 700:
Preço: 2.3

Produto: Feijão:
Estoque: 750:
Preço: 1.5
```

- **.keys()** : Um gerador com as chaves;
- **.values()** : Uma gerador com os valores;
- **.items()** : Uma gerador com os itens

Um desempacotamento é feito usando a chave.

16

Dicionários

Excluir uma entrada do dicionário com **del**.

```
>>> tabela = { "Alface": 0.45,
               "Batata": 1.20,
               "Tomate": 2.30,
               "Feijão": 1.50 }
>>> tabela
{'Tomate': 2.3, 'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
>>> del tabela["Tomate"]
>>> tabela
{'Batata': 1.2, 'Alface': 0.45, 'Feijão': 1.5}
```

17

CRUD

C R U D

Termos normalmente relacionado à Bancos de Dados.

18

CRUD em memória

CRUD (acrónimo do inglês **C**reate, **R**ead, **U**pdate e **D**elete) são quatro operações básicas (criação, consulta, atualização e destruição de dados) usadas em Bancos de Dados e interfaces de usuários.

Outros acrônimos podem ser usados para definir as mesmas operações:

CRUD:	ABCD:	VEIA:
Create (criar)	**A**dd (criar)	**V**er (ler)
Read (ler)	**B**rowse (ler)	**E**xcluir (remover)
Update (atualizar)	**C**hange (atualizar)	**I**nserir (criar)
Delete (remover)	**D**elete (remover)	**A**lterar (atualizar)

Podemos simular um CRUD em memória com estruturas de dados.

19

CRUD em memória

```
alunos = {}
```
dicionário vazio.

```
# create
matricula = 1
nome = 'aline'
nota = 9.0
aluno = (nome, nota)
alunos[matricula] = aluno
print('create: ', alunos)
create:  {1: ('aline', 9.0)}

# read:
matricula = 1
nome, nota = alunos[matricula]
print('read: ', nome, nota)
read:  aline 9.0

# update
matricula = 1
nome, nota = alunos[matricula]
nota = 9.5
alunos[matricula] = nome, nota
print('update: ', alunos)
update:  {1: ('aline', 9.5)}

# delete
matricula = 1
del alunos[matricula]
print('delete: ', alunos)
delete:  {}
```
dicionário vazio.

20