MC102 — Dicionários e Conjuntos

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-05-01 15:30

Motivando

Exercício:

1. Leia uma quantidade `n` de pares de nome/RA de alunos. Faça um programa que permita buscar o RA de um aluno a partir do seu nome.

Motivando

Um histograma é uma representação de dados utilizado na estatística. Simplificando, a ideia é exibir a quantidade de vezes que um certo dado aparece. Exemplo:

```
0: ***
1: *****
2: *
3:
4: ********
5: **
6: ****
7: *****
8:
9: **
```

Exercícios:

1. Faça um programa, que dado uma lista de número inteiros em $[0, 10)$, imprime um histograma para tais números.
2. Como modificar o programa anterior para fazer um histograma de nomes de alunos?

Dicionários

Listas armazenam os dados de acordo com os índices:

- Os dados são acessados pelo índice
  - Um `int`
- Para cada índice, temos um valor
  - É uma função no sentido matemático
- Podemos pensar que os nossos dados são pares índice-valor

Dicionários armazenam os dados em pares chave-valor:

- Os dados são acessados por uma chave (de busca)
  - Pode ser `str`, `int`, `float`
  - E até outros objetos (mas não qualquer objeto)
- Para cada chave, temos um valor
  - Também é uma função no sentido matemático
- Exemplos:
  - `ra["ana"] = 123456`
  - `nome[123456] = "ana"`
  - `d[3.2] = 10`

Criando um Dicionário

Dicionário vazio:

- `d = {}` (assim como `l = []`)
- `d = dict()` (assim como `l = list()`)

Dicionário com conteúdo inicial:

- Podemos passar os pares no formato `chave: valor`
  - `d = {"ana": 123456}`
  - `d = {"ana": 123456, "beto": 123457}`
- Ou usando o construtor `dict`
  - `d = dict([["ana", 123456], ["beto", 123457]])`
- Se todas as chaves forem strings, podemos fazer
  - `d = dict(ana=123456, beto=123457)`

As chaves de um dicionário podem ser de tipos diferentes

- Mas não temos chaves repetidas

Os valores também podem ser de tipos diferentes

- E podem ser repetidos

Leitura e Escrita

Para ler um valor a partir de uma chave, escrevemos:

- `dicionario[chave]`
- Ex: `print(ra["ana"])`
- Recebemos um `KeyError` se a chave não existe
- Podemos testar se a chave existe: `chave in dicionario`
  - Ex: `"ana" in ra`

Para escrever um valor a partir de uma chave, escrevemos:

- `dicionario[chave] = valor`
- `ra["ana"] = 123456`
- Podemos fazer isso mesmo se a chave não existir!
  - Essa é uma forma de adicionar pares no dicionário

Iterando

Sobre as chaves:

- basta usar `for chave in d:`
- ou `for chave in d.keys():`
- lembre-se que não há chave repetida

Sobre os valores:

- `for valor in d.values():`
- lembre-se que pode haver valor repetido

Sobre os pares:

- `for chave, valor in d.items():`

Importante: No Python 3.7 em diante a ordem de acesso das chaves é a ordem de inserção no dicionário

- Não é necessariamente verdade em versões anteriores

Outras informações

- Você pode apagar escrevendo `del dicionario[chave]`
- Você pode saber quantos pares há escrevendo `len(dicionario)`
- Você pode usar o método `get` ao invés das chaves para evitar `KeyError`
  - `d.get(chave, alt)`: se `chave` existir, devolve o item correspondente, senão devolve `alt`
- Há outros métodos também, veja a documentação!

Exercícios

1. Faça um programa que permita buscar o RA de um aluno a partir do seu nome.
2. Faça um programa, que dado uma lista de número inteiros em $[0, 10)$, imprime um histograma para tais números.
3. Repita o exercício anterior utilizando nomes de alunos.

Conjuntos

Conjuntos lembram dicionários:

- Porém, ao invés de armazenar pares chave-valor
- Armazenam apenas chaves
- As chaves podem ser `int`, `str`, `float`
- E até outros objetos... mas não todos...

Como na matemática, conjuntos não têm repetição

Criando conjuntos

Conjuntos com valores iniciais:

- `s = {"ana", "beto", "carlos"}`
  - Notação similar a do dicionário
  - Mas sem os valores, apenas chaves
- `s = set(["ana", "beto", "carlos"])`

Conjuntos vazios:

- `s = set()`
- Não podemos escrever `s = {}`
  - Porque isso cria um dicionário vazio!

Manipulando conjuntos

- `s.add(x)` adiciona `x`
  - apenas se `x` não estiver no conjunto
- `del` não funciona... (conjunto não tem chave)
- `s.discard(x)` remove `x`
  - Se `x` não está em `s`, nada acontece
- `s.remove(x)` remove `x`
  - Se `x` não está em `s`, temos um `KeyError`
- `x in s` para verificar se `x` está em `s`
- `for x in s:` para iterar sobre `s`
  - a ordem de acesso não é necessariamente a ordem de inserção no conjunto!

Exercícios

1. Faça uma função que, dados conjuntos `s1` e `s2`, devolve um novo conjunto representando a união de `s1` e `s2`
2. Faça uma função que, dados conjuntos `s1` e `s2`, devolve um novo conjunto representando a interseção de `s1` e `s2`
3. Faça uma função que, dados conjuntos `s1` e `s2`, devolve um novo conjunto representando a subtração de `s2` em `s1`
4. Faça uma função que, dados conjuntos `s1` e `s2`, devolve se `s1` é subconjunto de `s2`
5. Faça uma função que, dada uma lista, devolve uma nova lista sem elementos repetidos. Dica: use conjuntos!

Métodos úteis de conjunto

Devolvem um novo conjunto:

- `s1.union(s2)`: união
  - ou `s1 | s2`
  - O símbolo `|` representa **ou**
- `s1.intersection(s2)`: interseção
  - ou `s1 & s2`
  - O símbolo `&` representa **e**
- `s1.difference(s2, ...)`: diferença
  - ou `s1 - s2`
- `s1.symmetric_difference(s2)`: diferença simétrica
  - ou `s1 ^ s2`
  - O símbolo `^` representa **ou exclusivo**

Métodos úteis de conjunto

Alteram o próprio conjunto:

- `s1.update(s2)`: altera `s1` para a união de `s1` e `s2`
  - ou `s1 |= s2`
- `s1.intersection_update(s2)`: interseção
  - ou `s1 &= s2`
- `s1.difference_update(s2, ...)`: diferença
  - ou `s1 -= s2`
- `s1.symmetric_difference_update(s2)`: diferença simétrica
  - ou `s1 ^= s2`

Métodos úteis de conjunto

Relações:

- `s1.isdisjoint(s2)`: são disjuntos?
- `s1.issubset(s2)`: `s1` é subconjunto de `s2`?
  - ou `s1 <= s2`
- `s1.issuperset(s2)`: `s1` é superconjunto de `s2`?
  - ou `s1 >= s2`

Como sempre, veja a documentação para mais métodos e detalhes!