centro universitário FEI

CC8210 – NCA210 Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

Dicionários

Dicionários

- Dicionários (em Python) são **vetores associativos**
- Vetores associativos são coleções **desordenadas** de dados, usadas para **armazenar** valores como um **mapa**: por meio de elementos formados pelo par **chave** e **valor**
- Assim, diferentemente das listas, que contém um único valor como elemento, o dicionário contém o par: **chave:valor** (*key:value*)
  - **Chave (key):** serve para deixar o dicionário otimizado
  - **Valor (value):** valor do elemento associado a uma chave

Dicionários

- Dicionários diferem das listas essencialmente na maneira como os elementos são acessados:
  - **Listas**: valores são acessados por sua posição dentro da lista, via índice
  - **Dicionários**: valores são acessados por meio de suas **chaves** (*keys*)

Dicionários

- Um dicionário em Python funciona de forma semelhante ao dicionário de palavras:
  - As chaves (*keys*) de um dicionário devem ser exclusivas e com o tipo de dados imutáveis, como strings, inteiros ou tuplas
  - Porém, os valores (*values*) associados às chaves podem ser repetidos e de qualquer tipo

Criação dos Dicionários

- Para criarmos um dicionário, devemos incluir uma sequência de elementos dentro de chaves **{ }**, separados por vírgula.
- A chave e o valor são separados por dois pontos **:**
- Cada elemento do dicionário é um par composto por chave (*key*) e valor (*value*).
- Sintaxe:

```
d = {
    <key1>:<value1>,
    <key2>:<value1>,
    <key3>:<value2>
}
```

Exemplo

Criando um dicionário com chaves inteiras

```
dicionario = {
    1 : 'exemplo',
    2 : 'de',
    3 : 'dicionario'
}

print(dicionario)
```

```
{1: 'exemplo', 2: 'de', 3: 'dicionario'}
```

Sintaxe:

```
d = {
    <key1>:<value1>,
    <key2>:<value1>,
    <key3>:<value2>
}
```

Exemplo

Criando um dicionário com chaves de tipos mistos

```
teste = {
    'nome' : 'fulano',
    5 : 'cinco',
    'lista' : [1, 2, 4]
}

print(teste)
```

```
{'nome': 'fulano', 5: 'cinco', 'lista': [1, 2, 4]}
```

Acessando elementos

- Os valores são acessados por meio de suas chaves
- Utiliza-se o nome do dicionário e a chave dentro de colchetes [ ]

```
ingles = {
    'um' : 'one',
    'dois' : 'two',
    'tres' : 'three',
    'quatro' : 'four',
    'cinco' : 'five'
}
```

```
ingles['um']
```

```
'one'
```

```
ingles['quatro']
```

```
'four'
```

```
ingles_num = {
    1 : 'one',
    2 : 'two',
    3 : 'three',
    4 : 'four',
    5 : 'five'
}
```

```
ingles_num[3]
```

```
'three'
```

```
ingles_num[2]
```

```
'two'
```

Adicionando novos elementos

- Para adicionar um novo elemento a um dicionário existente, basta atribuir o novo valor e especificar a chave dentro de colchetes

```
ingles_num = {
    1 : 'one',
    2 : 'two',
    3 : 'three',
    4 : 'four',
    5 : 'five'
}
```

Dicionário original

```
ingles_num[7] = 'seven'
```

Acrescentando um novo elemento
chave = 7 e valor = ‘seven’

```
print(ingles_num)
```

```
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five', 7: 'seven'}
```

Removendo elementos

- Para remover um elemento de um dicionário utilizamos a palavra-chave **del**

Dicionário original

```
ingles_num = {
    1 : 'one',
    2 : 'two',
    3 : 'three',
    4 : 'four',
    5 : 'five'
}
```

Removendo o elemento pela chave = 3

```
del ingles_num[3]
```

```
print(ingles_num)
```

```
{1: 'one', 2: 'two', 4: 'four', 5: 'five'}
```

Removendo elementos

- Para remover um elemento de um dicionário utilizamos a palavra-chave **del**

Dicionário original

```
ingles = {
    'um' : 'one',
    'dois' : 'two',
    'tres' : 'three',
    'quatro' : 'four',
    'cinco' : 'five'
}
```

```
del ingles['quatro']
```

Removendo o elemento pela chave = 'quatro'

```
print(ingles)
```

```
{'um': 'one', 'dois': 'two', 'tres': 'three', 'cinco': 'five'}
```

Alguns métodos

- **items()**: retorna todos os elementos do dicionário - pares chave:valor

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}
```

```
d = dicionario.items()
```

```
d = list(d)
print(d)
```

```
[('um', 'exemplo'), ('dois', 'de'), ('tres', 'dicionario')]
```

Alguns métodos

- **keys()**: retorna todas as chaves do dicionário

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}
```

```
d = dicionario.keys()
```

```
d = list(d)
print(d)
```

```
['um', 'dois', 'tres']
```

```
print(d[0])
```

```
um
```

Alguns métodos

- **values()**: retorna todas valores do dicionário

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}
```

```
d = dicionario.values()
```

```
d = list(d)
print(d)
```

```
['exemplo', 'de', 'dicionario']
```

```
print(d[0])
```

```
exemplo
```

Mais alguns métodos

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Dicionários aninhados...

```
myfamily = {
    "child1" : {
        "name" : "Emil",
        "year" : 2004
    },
    "child2" : {
        "name" : "Tobias",
        "year" : 2007
    },
    "child3" : {
        "name" : "Linus",
        "year" : 2011
    }
}
```

Iteração

- Podemos utilizar estruturas de repetição para iterar por um dicionário

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}
```

Iterando pelas chaves

```
for chave in dicionario:
    print(chave)
```

```
um
dois
tres
```

Iterando pelos valores

```
for valor in dicionario.values():
    print(valor)
```

```
exemplo
de
dicionario
```

Iteração

- Podemos utilizar estruturas de repetição para iterar por um dicionário

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}
```

```
for chave, valor in dicionario.items():
    print(chave, valor)
```

```
um exemplo
dois de
tres dicionario
```

Existência

- Para determinar se uma chave especifica está presente em um dicionário use a palavra-chave **in**.
- Exemplo:

```
dicionario = {
    'um' : 'exemplo',
    'dois' : 'de',
    'tres' : 'dicionario'
}

if 'dois' in dicionario:
    print("Sim, 'dois' é uma das chaves do dicionario")
```

```
Sim, 'dois' é uma das chaves do dicionario
```

Exemplo

Crie um programa que imprime o número de caracteres únicos em uma *string* criada pelo usuário. Por exemplo, *Hello, World!* tem 10 caracteres únicos, enquanto *zzz* tem somente 1 caractere único. Use um dicionário para resolver este problema.

Exemplo

```
letras = {}
texto = input("Insira um texto qualquer: ")

for x in texto:
    letras[x] = 0

print(letras)
print("%d caracteres únicos" % len(letras))
```

```
Insira um texto qualquer: Hello, World!
{'H': 0, 'e': 0, 'l': 0, 'o': 0, ',': 0, ' ': 0, 'W': 0, 'r': 0, 'd': 0, '!': 0}
10 caracteres únicos
```

Exemplo

Crie um programa que imprime os caracteres únicos com as respectivas quantidades em uma *string* recebida como parâmetros, imprima um dicionário, sendo os keys as letras e os values as quantidades.
Por exemplo:

Hello, World!
{'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 1, 'W': 1, 'r': 1, 'd': 1, '!': 1}

zzz tem somente 1 caractere único.
{'z': 3}

Exemplo

```
letras = {}
texto = input("Insira um texto qualquer: ")

for x in texto:
    if x in letras:
        letras[x] += 1
    else:
        letras[x] = 1

print(letras)
```

```
Insira um texto qualquer: Hello, World!
{'H': 1, 'e': 1, 'l': 3, 'o': 2, ',': 1, ' ': 1, 'W': 1, 'r': 1, 'd': 1, '!': 1}
```

Dicionário de Funções

- É possível usar um dicionário para guardar funções a serem executadas.

Dicionário de Funções

```
d = {
    '+': lambda x,y: x+y,
    '-': lambda x,y: x-y,
    '*': lambda x,y: x*y,
    '/': lambda x,y: x/y
    }

op = '+'
num1 = 2
num2 = 5

try:
    print(d[op](num1, num2))
except KeyError:
    print("error")
```

```
7
```

Dicionários e Arquivos

- Dicionários são muito interessante para armazenar dados de maneira organizada no Python.
- Mas:
  - Ao terminar o programa, o dicionário desaparece.
- Solução:
  - Salvar o dicionário em um arquivo.
  - Diversas maneiras:
    - String, JSON, NumPy, Pandas, ...

Salvando um Dicionário em um arquivo texto

```
def save_dict_to_file(dic):
    f = open('dict.txt','w')
    f.write(str(dic))
    f.close()
def load_dict_from_file():
    f = open('dict.txt','r')
    data=f.read()
    f.close()
    return eval(data)
```

```
dicionario = {
    1 : 'exemplo',
    2: 'de',
    3: 'dicionario'
}
print (dicionario)
save_dict_to_file(dicionario)
dicionario2 = load_dict_from_file()
print (dicionario2)
```

```
{1: 'exemplo', 2: 'de', 3: 'dicionario'}
{1: 'exemplo', 2: 'de', 3: 'dicionario'}
```

Tuplas

Tuplas

- Tuplas são similares às listas, porém são **imutáveis**!
- Tuplas não permitem adicionar, apagar, inserir ou modificar elementos
- **Tuplas são definidas com parêntesis ( )**; **Listas com colchetes [ ]**
- Exemplo:

```
b = (2, 4, 5, 'tupla')
print( b )
```

```
(2, 4, 5, 'tupla')
```

```
print( b[1] )
```

```
4
```

Tuplas

- Tuplas são imutáveis!
- Exemplo:

```
b[1] = 8
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
 last)
<ipython-input-2-937fcf8f2e75> in <module>
      1 b = (2, 4, 5, 'tupla')
      2
----> 3 b[1] = 8

TypeError: 'tuple' object does not support item assignment
```

Conjuntos ou Sets

Conjuntos ou Sets

- Um conjunto é uma coleção de valores distintos:
- Pode-se implementar conjuntos de diversas formas:
  - Uma lista de valores:
    - É necessário tomar o cuidado para evitar valores duplicados.
  - Um dicionário:
    - As chaves de um dicionário são necessariamente únicas.
    - O valor associado a cada chave pode ser qualquer um.
- Existe em Python um tipo primitivo que implementa conjuntos.
  - Mais apropriado do que o uso de listas ou dicionários

O tipo set

- Pode-se construir um conjunto usando a construção:
  `set(`*`sequencia`*`)`
- Onde *`sequência`* é uma sequencia qualquer, como uma lista, uma tupla ou uma string.
  - Caso seja uma lista, os elementos devem ser imutáveis
- Exemplo:
  - `set((1,2,3))`
  - `set ("xxabc")`

Exemplos

```
set((1,2,3))
Out[15]: {1, 2, 3}

set ("xxabc")
Out[16]: {'a', 'b', 'c', 'x'}

set ([1,[1,2],3,1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-12612d18feb4> in <module>
      5 Out[16]: {'a', 'b', 'c', 'x'}
      6
----> 7 set ([1,[1,2],3,1])

TypeError: unhashable type: 'list'
```

Ireração nos conjuntos

- Pode-se também usar o comando for com sets
- Observe que a iteração não necessariamente visita os elementos na mesma ordem em que eles foram inseridos no conjunto:

```
s = set([1,2,9,100,"a"])
for x in s:
    print (x)
```

```
1
2
100
9
a
```

Por que a ordem muda?

- Um conjunto é uma estrutura de dados otimizada para operações de conjunto.
- Como um conjunto matemático, ele não impõe ou mantém qualquer ordem particular dos elementos.
  - O conceito abstrato de um conjunto não impõe ordem, então a implementação não é necessária.
- Quando você cria um conjunto a partir de uma lista, o Python tem a liberdade de alterar a ordem dos elementos, organizando de uma forma altamente otimizada, assim realizando as operações de conjunto com eficiência.

Sets

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

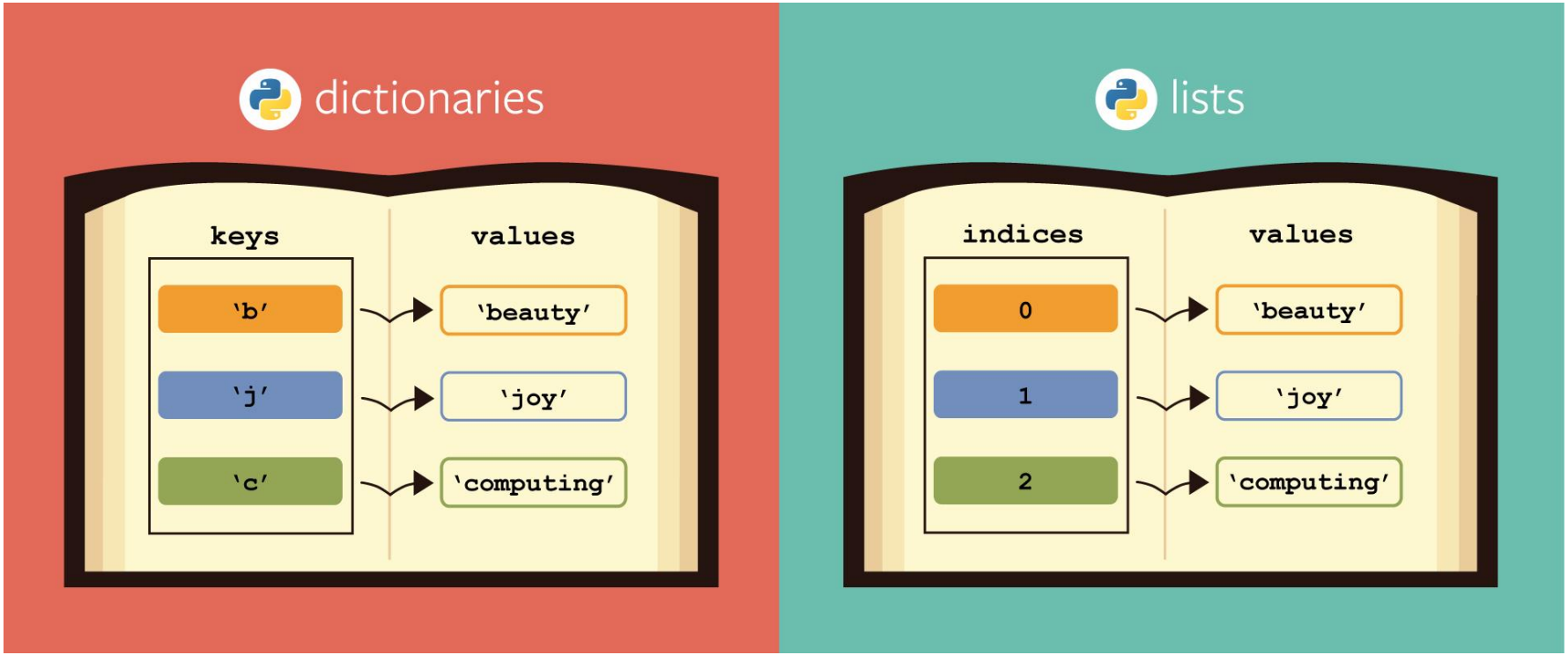
Conclusão

Conclusão

- Dicionários provém uma maneira simples de manter os dados organizados de maneira eficiente no computador.
- Tuplas são imutáveis
- E existem conjuntos no Python...

- Em Python, tudo é muito simples.

Fim


dictionaries
keys
values
'b'
'beauty'
'j'
'joy'
'c'
'computing'
lists
indices
values
0
'beauty'
1
'joy'
2
'computing'

Sites usados

- https://www.w3schools.com/python/python_dictionaries.asp
- Slides de Claudio Esperança, UFRJ:
  - http://orion.lcg.ufrj.br/python/_11%20-%20Programando%20em%20Python%20-%20Conjuntos.pdf