Computação II - Python
Aula 4 - Estrutura de Dados - Dicionários e Conjuntos

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ

Classes e Objetos na programação

A classe lista e a classe string

- Você não sabia disso, porém os tipos de dados lista e string que vimos anteriormente são na verdade a *classes*.

```
In [1]:  L=[]

In [2]: s="abcde"

In [3]: type(L)
Out[3]: <class 'list'>

In [4]: type(s)
Out[4]: <class 'str'>
```

Classes e Objetos na programação

A classe lista e a classe string

- As funções *list.* <*nome da função*>*(lista, outros parametros)* e *str.* <*nome da função*>*(string, outros parametros)* são na verdade definidas como métodos respectivamente das classes lista e string, e a maneira mais usual de utilizá-las é fazendo uso da notação de método de uma classe.
- Primeiro colocamos o nome da lista ou da string, depois o ponto, depois o nome do método e não precisamos passar a lista ou a string desejada como parâmetro, pois o objeto para o qual o método está sendo chamado já foi indicado.

```
1
2 In [5]: L.append(1)
3
4 In [6]: L
5 Out[6]: [1]
6
7 In [7]: s.replace("a","1")
8 Out[7]: '1bc'
```

- É o parâmetro *self* que está na definição de todos os métodos de uma classe que nos desobriga de passar, na hora de chamar o método, o objeto lista ou string desejado como parâmetro.

Dicionário

Considere que você precisa fazer uma função que guarde o nome e o telefone de seus amigos. Sua função também deve permitir a consulta aos telefones das pessoas.

Como guardar estas informações (nome e telefone)?

Como recuperar o telefone de uma dada pessoa?

Dicionário

Dicionários são estruturas para armazenar dados. Não são sequências como strings, listas e tuplas.

São **mapeamentos** formados por pares de *chave* – *valor*.

$$\textit{Chave}1 \rightarrow \text{Conteúdo1}$$
$$\textit{Chave}2 \rightarrow \text{Conteúdo2}$$
$$\textit{Chave}3 \rightarrow \text{Conteúdo3}$$
$$\ldots$$

Representam uma coleção não ordenada de **valores** onde cada valor é referenciado através de sua **chave**.

Notação: { chave1: conteúdo1, chave2: conteúdo2, ..., chaveN: conteúdoN }

Dicionário

Dicionários são **mapeamentos** formados por pares de *chave* − *valor*.

As **chaves** funcionam como os índices de uma lista

$$Chave1 \rightarrow \text{Conteúdo1}$$
$$Chave2 \rightarrow \text{Conteúdo2}$$
$$Chave3 \rightarrow \text{Conteúdo3}$$
$$\ldots$$

As **chaves de dicionários** são dados de tipo imutável, geralmente strings (podem ser tuplas ou tipos numéricos).

Os valores em um dicionário são dados quaisquer.

Dicionário

```
In [1]: Caderno = {"Carlos": "2222-2223", "Andre": "2121-9092","Jose": "9999-9291"}

In [2]: Caderno["Andre"]
Out[2]: '2121-9092'

In [3]: Caderno["Jorge"]
Traceback (most recent call last):
File "<ipython-input-3-0a51cd176d67>", line 1, in <module>
Caderno["Jorge"]
KeyError: 'Jorge'

In [4]: "Jorge" in Caderno
Out[4]: False

In [5]: len(Caderno)
Out[5]: 3
```

Dicionário

Podemos usar a função **dict** para definir dicionários:

- **Lista de pares (chave,valor):**

  Caderno = dict([("Carlos","2222-2223"),("André","2121-9992"),
  ("José","9999-9291")])

- **Sequência de itens no formato chave=valor:**

  Caderno = dict(Andre="2121-9992", Jose="9999-9291",
  Carlos="2222-2223")

Dicionário

Podemos usar list comprehensions para criar dicionários:

```
In [1]: d = {x: x**2 for x in (2, 4, 6)}

In [2]: d
Out[2]: {2: 4, 4: 16, 6: 36}
```

Dicionário

```
In [1]: Caderno = {"Carlos": "2222-2223","Andre":"2121-9092","Jose": "9999-9291"}

In [2]: Caderno["Jose"]
Out[2]: '9999-9291'

In [3]: Caderno["Jose"] = "8799-0405"

In [4]: Caderno["Jose"]
Out[4]: '8799-0405'
```

Dicionários são mutáveis. Mesma sintaxe da inserção! Não é inserida outra chave com o mesmo nome e valor diferente, pois chaves são únicas.

Dicionário

```
In [1]: Caderno = {"Carlos":"2222-2223","Andre":"2121-9092","Jose":"9999-9291"}

In [2]: "Jorge" in Caderno
Out[2]: False

In [3]: Caderno["Jorge"] = "8586-9091"

In [4]: Caderno["Jorge"]
Out[4]: '8586-9091'

In [5]: Caderno
Out[5]: {'Andre':'2121-9092','Jorge':'8586-9091','Jose':'8799-0405',
'Carlos':'2222-2223'}
```

- **Para adicionar um novo par chave:valor**
  Perceba que, diferentemente de listas, atribuir a um elemento de um dicionário não requer que uma posição exista previamente.
- O dicionário não fornece garantia de que as chaves estarão ordenadas, mas a ordem em que os elementos aparecem não é importante, pois os valores são acessados somente através de suas respectivas chaves, e não de suas posições.

Manipulação de Dicionário

Como saber quais são as chaves e os valores de um dicionário **d**?

- **d.keys()**: retorna uma nova visão das chaves do dicionário.
- **d.values()**: retorna uma nova visão dos valores do dicionário.

```
In [1]: meses = {"janeiro":31,"fevereiro":28,"marco":31,"abril":30,"maio":31,
"junho":30,"julho":31,"agosto":31,"setembro":30,"outubro":31,"novembro":31,
"dezembro":31}

In [2]: meses.keys()
Out[2]: dict_keys(['janeiro', 'fevereiro', 'marco', 'abril', 'maio', 'junho', 'julho', '
     agosto', 'setembro', 'outubro', 'novembro', 'dezembro'])

In [3]: list(meses.keys())
Out[3]: ['janeiro','fevereiro','marco','abril','maio','junho','julho','agosto','setembro'
     , 'outubro','novembro','dezembro']

In [4]: meses.values()
Out[4]: dict_values([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31])

In [5]: list(meses.values())
Out[5]: [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31]
```

Manipulação de Dicionário

- **d.items()**: retorna uma nova visão dos itens do dicionário (pares (chave, conteúdo)).

```
In [1]: meses = {"janeiro":31, "fevereiro":28, "marco":31, "abril":30, "maio":31,
"junho":30, "julho":31, "agosto":31, "setembro":30,"outubro":31,
"novembro":31,"dezembro":31}

In [2]: meses.items()
Out[2]: dict_items([('janeiro', 31), ('fevereiro', 28), ('marco', 31), ('abril', 30), ('
    maio', 31), ('junho', 30), ('julho', 31), ('agosto', 31), ('setembro', 30), ('
    outubro', 31), ('novembro', 31), ('dezembro', 31)])

In [3]: list(meses.items())
Out[3]: [('janeiro', 31),('fevereiro', 28),('marco', 31),('abril', 30),('maio', 31),('
    junho', 30),('julho', 31),('agosto', 31),('setembro', 30),('outubro', 31),('novembro
    ', 31),('dezembro', 31)]
```

Manipulação de Dicionário

- **d.get(chave,[valor de retorno])**: Retorna o valor associado com a chave.
  Se *chave* não está no dicionário e se o *valor de retorno* é especificado, retorna o valor especificado.
  Se o *valor de retorno* não é especificado, retorna **None**.

```
In [1]: meses = {"janeiro":31, "fevereiro":28, "marco":31, "abril":30,"maio":31,
"junho":30, "julho":31, "agosto":31, "setembro":30, "outubro":31, "novembro":31,
"dezembro":31}

In [2]: meses.get("marco")
Out[2]: 31

In [3]: meses.get("marco","Valor nao encontrado")
Out[3]: 31

In [4]: meses.get("Marco")
Out[4]:

In [5]: meses.get("Marco","Valor nao encontrado")
Out[5]: "Valor nao encontrado"
```

Manipulação de Dicionário

- **d.clear()**: apaga todos os itens do dicionário.
- **d.copy()**: cria e retorna uma cópia do dicionário.

```
In [1]: meses = {"janeiro":31, "fevereiro":28}

In [2]: novo = meses.copy()

In [3]: novo
Out[3]: {'fevereiro': 28, 'janeiro': 31}

In [4]: novo["maio"] = 31

In [5]: novo
Out[5]: {'fevereiro': 28, 'janeiro': 31, 'maio': 31}

In [6]: meses
Out[6]: {'fevereiro': 28, 'janeiro': 31}

In [7]: meses.clear()

In [8]: meses
Out[8]: {}

In [9]: novo
Out[9]: {'fevereiro': 28, 'janeiro': 31}
```

Manipulação de Dicionário

- **d.copy()**: cria e retorna uma cópia do dicionário. Os elementos mutáveis (listas ou dicionários) do novo dicionário são apenas referências aos elementos do dicionário original.

```
In [1]: meses = {"janeiro":30, "fevereiro":[28,29]}
In [2]: novo = meses.copy()
In [3]: novo
Out[3]: {'fevereiro': [28,29], 'janeiro': 30}
In [4]: novo["maio"]=31
In [5]: novo
Out[5]: {'fevereiro': [28, 29], 'janeiro': 30, 'maio': 31}
In [6]: meses
Out[6]: {'fevereiro': [28,29], 'janeiro': 30}
In [7]: novo["janeiro"]=31
In [8]: novo
Out[8]: {'fevereiro': [28, 29], 'janeiro': 31, 'maio': 31}
In [9]: meses
Out[9]: {'fevereiro': [28,29], 'janeiro': 30}
In [10]: novo["fevereiro"].pop()
Out[10]: 29
In [11]: novo
Out[11]: {'fevereiro': [28], 'janeiro': 31, 'maio': 31}
In [12]: meses
Out[12]: {'fevereiro': [28], 'janeiro': 30}
```

Manipulação de Dicionário

Escreva uma função que receba uma frase como parâmetro e retorne um dicionário, onde cada chave seja um caracter e seu valor seja o número de vezes que o caracter aparece na frase lida.
Exemplo: “Os ratos” $\rightarrow$ {“O”:1, “ “:1, “s”:2, “r”:1, “a”:1, “t”:1,”o”:1}
Faça uma versão usando while e outra usando for.

Manipulação de Dicionário

Escreva uma função que receba uma frase como parâmetro e retorne um dicionário, onde cada chave seja um caracter e seu valor seja o número de vezes que o caracter aparece na frase lida.
Exemplo: "Os ratos" $\rightarrow$ {"O":1, " ":1, "s":2, "r":1, "a":1, "t":1,"o":1}
Faça uma versão usando while e outra usando for.

```
1  def cria dicionario(frase):
2
3  """Solucao com while
4  Parametro de entrada: str
5  Valor de retorno: dic"""
6
7    dicionario = { }
8    i=0
9    while i < len(frase):
10     if frase[i] not in dicionario:
11       dicionario[frase[i]] = 1
12     else:
13       dicionario[frase[i]] += 1
14     i += 1
15   return dicionario
```

```
1  def cria dicionario(frase):
2
3  """Solucao com for
4  Parametro de entrada: str
5  Valor de retorno: dic"""
6
7    dicionario = { }
8    for c in frase:
9      if c not in dicionario:
10       dicionario[c] = 1
11     else:
12       dicionario[c] += 1
13   return dicionario
```

Manipulação de Dicionário

1. Escreva uma função que transforma uma lista de tuplas em um dicionário que associa o primeiro componente de cada tupla ao segundo.

2. Escreva uma função que transforma um dicionário em uma lista de tuplas, onde as tuplas estão ordenadas pelo primeiro componente. Lembre-se do método **sort** para listas.

Conjuntos

Conjuntos são coleções **não ordenadas** de elementos **sem repetições**.

```
1
2  In [1]: s1 = {'a', 'b'}
3
4  In [2]: type(s1)
5  Out[2]: set
6
7  In [3]: s2 = {'b', 'c', 'b'}
8
9  In [4]: s2
10 Out[4]: {'b', 'c'}
```

Conjuntos

As operações usuais de conjuntos estão definidas para dados do tipo **set**

```
In [4]: s2
Out[4]: {'b', 'c'}

In [5]: s1
Out[5]: {'a', 'b'}

In [6]: s1.issubset(s2)
Out[6]: False

In [7]: s3 = {'a'}

In [8]: s3.issubset(s1)
Out[8]: True

In [9]: s1.intersection(s2)
Out[9]: {'b'}

In [10]:'b' in s1
Out[10]: True
```

Conjuntos

A função **set()** cria um conjunto a partir de uma sequência passada como parâmetro.

```
In [1]: s4 = set((1,2,3))

In [2]: s4
Out[2]: {1,2,3}

In [3]: a = set('abracadabra')

In [4]: b = set('alacazam')

In [5]: a
Out[5]: {'a', 'r', 'b', 'c', 'd'}
```

Conjuntos

A função **set()** cria um conjunto a partir de uma sequência passada como parâmetro.

```
In [5]: a
Out[5]: {'a', 'r', 'b', 'c', 'd'}

In [6]: b
Out[6]: {'a', 'l', 'k', 'z', 'm'}

In [7]: a - b   # elementos em a que nao estao em b
Out[7]: {'r', 'd', 'b'}

In [8]: a | b           # elementos em a uniao b
Out[8]: {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

In [9]: a & b           # elementos na interseccao de a e b
Out[9]: {'a', 'c'}

In [10]: a ^ b          # elementos em a ou em b mas nao em ambos
Out[10]: {'r', 'd', 'b', 'm', 'z', 'l'}
```

Conjuntos: set comprehensions

Assim como list comprehensions, temos também **set comprehensions**:

```
In [1]: a = {x for x in 'abracadabra' if x not in 'abc'}

In [2]: a
Out[2]: {'d', 'r'}
```

Conjuntos

1. *(Exercício 4 da seção Python Essentials do site Quantecom)* Escreva uma função que receba como entrada duas sequencias seqA e seqB e retorne True se todo elemento de seqA for também elemento de seqB, caso contrário retorne False. Use o tipo de dado **set** e as operações de conjunto.

2. Repita o exercício anterior **sem** utilizar o tipo set e as operações de conjunto.

Computação II - Python
Aula 4 - Estrutura de Dados - Dicionários e Conjuntos

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ