



# Algoritmos Computacionais

**Conteúdo:** Conjuntos e dicionários

Prof. Dsc. Giomar Sequeiros  
[giomar@eng.uerj.br](mailto:giomar@eng.uerj.br)

# Conjuntos

# Conjuntos

- São coleções não-ordenadas de objetos simples
- São usados quando a existência de um objeto em uma coleção é mais importante do que a ordem ou o número de vezes que ocorre
- Pode-se implementar conjuntos usando lista, por exemplo, mas deve-se tomar o cuidado de evitar valores duplicados
- Python suporta o tipo primitivo chamado set, que implementa conjuntos
  - Mais apropriado do que usar lista com esse propósito
- Exemplo:

```
>>> A = {1, 2, 3, 4, 5}
>>> type(A)
<class 'set'>
```

# O tipo set

- Pode-se construir um set usando a construção `set(sequencia)`
  - Onde sequência é uma sequência qualquer, como uma lista, uma tupla, ou uma string
  - Caso uma lista seja utilizada, os elementos devem ser imutáveis
- Exemplos:

```
>>> set((1,2,3))
{1, 2, 3}
>>> set("xxabc")
{'x', 'c', 'a', 'b'}
>>> set([1,(1,2),3,1])
{3, 1, (1, 2)}
>>> set([1,[1,2],3,1])
```

ERRO! O elemento [1,2] é uma lista. Lista são mutáveis

# O tipo set

---

Para criar Sets vazios podemos fazer:

```
s = set()
```

```
s = set([])
```

```
s = set("")
```

# Trabalhando com sets

---

- **`x in s`** → True se o elemento `x` pertence a `s`
- **`s.add(x)`** → Inclui o elemento `x` em `s`
- **`a = s.copy()`** → Retorna uma cópia de `s` em `a`
- **`s.union(r)`** → Retorna a união entre `s` e `r`
- **`s.intersection(r)`** → Retorna a interseção entre `s` e `r`
- **`s.difference(r)`** → Retorna a diferença entre `s` e `r`
- **`list(s)`** → Retorna os elementos de `s` numa lista
- **`tuple(s)`** → Retorna os elementos de `s` numa tupla

# Exemplos

```
>>> s = {1,2,3}
>>> r = {2,5,9,1}
>>> 1 in s
True
>>> 1 in r
True
>>> 3 in r
False
>>> s.union(r)
{1, 2, 3, 9, 5}
>>> s.intersection(r)
{1, 2}
>>> s.difference(r)
{3}
>>> r.difference(s)
{9, 5}
>>> s.add(5)
>>> s.intersection(r)
{1, 2, 5}
```

# Iterando sobre sets

- Pode-se também usar o comando FOR com sets
- Diferente de uma lista, set é uma estrutura não-ordenada. Devido a essa característica, a iteração não necessariamente visita os elementos na mesma ordem em que eles foram inseridos no conjunto
- Exemplo:

```
A = {1, 2, 9, 100, 'a'}
```

```
for x in A:  
    print(x)
```



# Outros métodos

---

- **s.discard(x)** → Exclui o elemento x de s (se existir)
- **s.remove(x)** → Exclui o elemento x de s (se não existir dá erro)
- **s.issubset(r)** → True sse s contido em r
- **s.issuperset(r)** → True sse s contém r
- **s.symmetric\_difference(r)** → Retorna a diferença simétrica entre s e r, isto é, a união entre s e r menos a interseção de s e r
- **s.update(r)** → Mesmo que  $s = s.union(r)$
- **s.intersection\_update(r)** → Mesmo que  $s = s.intersection(r)$
- **s.difference\_update(r)** → Mesmo que  $s = s.difference(r)$

# Exemplos

```
>>> s = set([1,2,3])
>>> r = set([2,5,9])
>>> s.issuperset(r)
False
>>> s.update(r)
>>> s
set([1, 2, 3, 5, 9])
>>> s.issuperset(r)
True
>>> r.issubset(s)
True
>>> s.discard(5)
>>> s
set([1, 2, 3, 9])
>>> s.symmetric_difference(r)
set([3, 5, 1])
```

# Exercícios

---

- 1) Escreva um programa que deve receber vários nomes completos de pessoas. O critério de parada é receber um nome vazio. Em seguida o programa deve imprimir apenas os primeiros nomes de cada pessoa, sem repetição.
- 2) Escreva um programa que recebe duas listas de valores e verifica se uma lista é subconjunto da outra. O programa deve imprimir qual lista está contida na outra ou informar que não há relação entre elas.
- 3) Em um jantar foram servidas duas sobremesas. Das pessoas presentes no jantar, algumas comeram a sobremesa X, outras comeram a sobremesa Y, algumas comeram as duas e algumas não comeram nenhuma das duas sobremesas. Desenvolva um programa que receba 3 listas, sendo elas: nomes dos convidados, nomes de quem comeu a sobremesa X, nomes de quem comeu a sobremesa Y. Seu programa deve imprimir o nome dos convidados que não comeram nenhuma das duas sobremesas.

# Dicionários

# Definição

- Dicionários são estruturas para armazenar dados, mas **NÃO** são **indexados sequencialmente** da forma que listas, strings e tuplas
- Seus elementos são representados por pares (**chave, valor**)
- Seu sistema de **endereçamento** é por **chaves**. Cada chave tem um **valor** atribuído. Se você quer saber um valor, deve perguntar pela sua chave associada.

Chave	Valor
'Ze'	'30/06'
'Maria'	'24/09'
'Joao'	'08/02'

# Sintaxe

- A **chave** precisa ser de um tipo imutável
- Geralmente são strings, mas podem ser tuplas ou tipos numéricos
- O **valor** pode ser tanto mutável quanto imutável

dicionario = {chave1: valor1, chave2: valor2, ...}

- Exemplo:

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000}
```

- Equivalente a:

```
>>> ficha = {'profissão': 'engenheiro', 'salario': 7000, 'nome': 'Carlos' }
```

- Em um dicionário não importa a ordem das chaves, pois os valores NÃO são acessados a partir de suas posições.

# Sintaxe

---

- Para adicionar um novo par chave-valor:

```
>>> ficha['idade'] = 34
```

- Para recuperar o valor atribuído a uma chave:

```
>>> ficha['idade']  
34
```

- Dicionários são mutáveis. Para alterar o valor contido em uma chave:

```
>>> ficha['salario'] = 7200  
>>> ficha['salario']  
7200
```

- É a mesma sintaxe da inserção! Não é inserida uma chave nova, mas sim atualizada a existente. Chaves são únicas.

# Métodos úteis: pop

---

## pop(chave)

- Obtém o valor correspondente à chave e remove o par chavevalor do dicionário

```
>>> ficha.pop('salario')
7200
>> ficha
{'profissão ': 'engenheiro', 'idade': 34, 'nome': 'Carlos'}
```



# Métodos úteis: del e clear

## del

- O comando **del** também pode ser usado para remover alguma chave do dicionário

```
>>> ficha = {'nome': 'Carlos', 'idade': 34}
>>> del ficha['idade']
>>> ficha
{'nome': 'Carlos'}
```

## clear():

- Remove todos os elementos do dicionário

```
>>> ficha.clear()
>>> ficha
{}
```

# Métodos úteis: copy

## copy()

- Cria e retorna um dicionário cópia com os mesmos pares chave-valor do original

```
>>> x = {'Joao': [1, 2], 'Maria': [3, 4]}
>>> y = x.copy()
>>> y ['Pedro'] = [5, 6]
>>> x ['Joao'] += [3]
>>> x
{'Joao': [1, 2, 3], 'Maria': [3, 4]}
>>> y
{'Pedro': [5, 6], 'Joao': [1, 2], 'Maria': [3, 4]}
```

# Métodos úteis: update

## Update(dic)

Atualiza um dicionário com os elementos de outro. Os itens são adicionados um a um ao dicionário original.

```
>>> x = {'a':1, 'b':2, 'c':3}
>>> y = {'z':9, 'b':7}
>>> x.update(y)
>>> x
{'a': 1, 'c': 3, 'b': 7, 'z': 9}
>>> x.update(a=7, c='novo')
>>> x
{'a': 7, 'c': 'novo', 'b': 7, 'z': 9}
```

# Métodos úteis: has\_key

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000 ,  
'idade': 34}
```

## has\_key(chave)

Pergunta se o dicionário possui a chave passada como parâmetro

```
>>> ficha.has_key('idade')  
True  
>>>  
ficha.has_key('endereco')  
False
```

# Métodos úteis: keys

---

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000 ,  
'idade': 34}
```

## keys()

Retorna as chaves do dicionário

```
>>> list(ficha.keys())  
['idade', 'salario', 'profissao', 'nome']
```

# Métodos úteis

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro',  
'salario': 7000, 'idade': 34}
```

**values():** Retorna quais valores estão contidos no dicionário

```
>>> list(ficha.values())  
[34, 7200, 'engenheiro', 'Carlos']
```

**items():** Retorna uma lista de tuplas contendo os pares (chave, valor)

```
>>> list(ficha.items())  
[('idade', 34), ('salario', 7200), ('profissao', 'engenheiro'),  
( 'nome', 'Carlos')]
```

# Iteração em Dicionários

Uma das maneiras de iterar sobre um dicionário é utilizando o `for` e o método `keys()`. Cada elemento é interpretado como sendo a chave do dicionário.

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000, 'idade': 34}

>>> for elemento in ficha.keys():
    print(elemento)
```

**Saída:**

```
idade
salario
profissao
nome
```

# Iteração em Dicionários

Como a variável de controle do for assume o valor das chaves do dicionário, podemos acessar o valor por meio dessa variável, conforme o exemplo a seguir:

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000, 'idade': 34}

>>> for elemento in ficha.keys():
    print(elemento, ': ', ficha[elemento])
```

**Saída:**

```
idade : 34
salario : 7200
profissao : engenheiro
nome : Carlos
```



# Iteração em Dicionários

Também podemos utilizar o método `items()`. Para isso devemos lembrar que:

- O método `items()` retorna uma lista de tuplas contendo o par (chave,valor)
- O `for` pode vasculhar qualquer sequência, inclusive uma lista de tuplas

```
>>> ficha = {'nome': 'Carlos', 'profissão': 'engenheiro', 'salario': 7000, 'idade': 34}
```

```
>>> for tupla in ficha.items():  
    print(tupla[0], ': ', tupla[1])
```

ou

```
>>> for a,b in ficha.items():  
    print(a, ': ', b)
```

Saída:

```
idade : 34  
salario : 7200  
profissao : engenheiro  
nome : Carlos
```

# Exercício

---

1. Crie uma função chamada "calcula\_media" que recebe um dicionário contendo notas de um aluno, onde as chaves são os nomes das disciplinas e os valores são as notas. A função deve retornar a média das notas.

Exemplo de entrada:

```
notas_aluno = {"Matemática": 8.5, "Ciências": 7.2, "História": 6.8}
```

Saída: 7.5

2. Crie uma função chamada "maior\_nota" que recebe um dicionário contendo notas de um aluno e retorna a disciplina em que o aluno obteve a maior nota