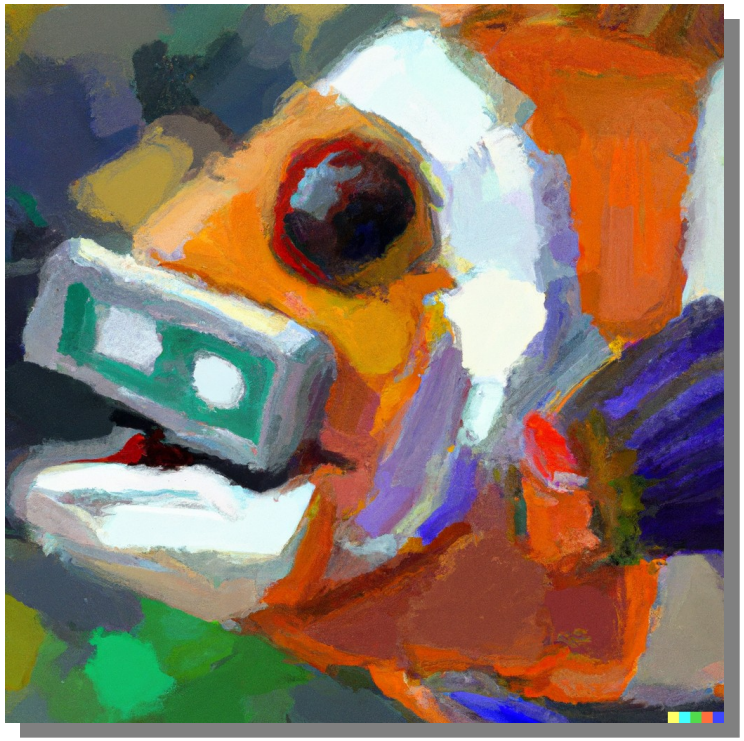
Universidade de Brasília
Departamento de Ciência da Computação



Semana 11

Listas (parte 1)

CIC0004
Algoritmos e Programação de Computadores

Prof. Pedro Garcia Freitas https://pedrogarcia.gitlab.io/ pedro.garcia@unb.br

Brasilia

1. Objetivos

Esta aula introduz o conceito de coleções (listas e tuplas) e demonstrar como esse conceito é usável em linguagem Python.

2. Introdução

- Listas são sequências ordenadas de elementos
  - Separadas por vírgula
  - Entre colchetes ("[…]")
- Exemplos:
  - **[1, 2, 3,4 ,5]**
  - **['Alice', 'Bob', 'Carol', Dean']**
  - **['Alice', 'freshman', [100, 90, 70]]**

3. Usos

- Agrupar objetos relacionados:
  - Lista de pessoas, de inteiros, de exames, etc
  - Vetores de vetores (matrizes) de álgebra linear
- Armazenar registros
  - Entrada de uma lista de contactos, por exemplo:
    - **['Bill', '713-555-1234', 'bill.tran@ xyx.com']**

4. Listas & Strings

Listas

- Sequências ordednadas
  - Primitivas similares
- Elementos da lista podem ser objetos de qualquer tipo
  - Incluindo listas
- São mutáveis (*side effects*)

Strings

- Sequências ordenadas
  - Primitivas similares (char)
- Só podem conter caracteres
- Não-mutáveis.

5. Operações

- Encontrar o tamanho de uma lista
- Acessar os elementos de uma lista
- Decidir se há a pertinência numa lista
- Concatenar 2 ou mais listas
- Acessar fatias da lista (*slices*)
- Modificar elementos individuais
  - Listas são **mutáveis**

6. Accessando elementos de uma lista

```
>>> names = ['Ann', 'Bob', 'Carol', 'end']
>>> names
['Ann', 'Bob', 'Carol', 'end']
>>> print(names)
['Ann', 'Bob', 'Carol', 'end']
>>> names[0]
'Ann'
```

- *Podemos acessar os elementos individualmente pelo seu índice*

6. Accessando elementos de uma lista

```
>>> names=['Ann', 'Bob', 'Carol', 'end']
>>> names[1]
'Bob'
>>> names[-1]
'end'
```

- *A indexação de listas usa as mesmas convenções que a indexação de strings.*

7. Tamanho de uma lista

```
names = ['Ann', 'Bob', 'Carol', 'end']
>>> len(names)
4
>>>alist = ['Ann', 'Bob', 'Carol',
 [1,2,3]]
>>> len(alist)
 4
```

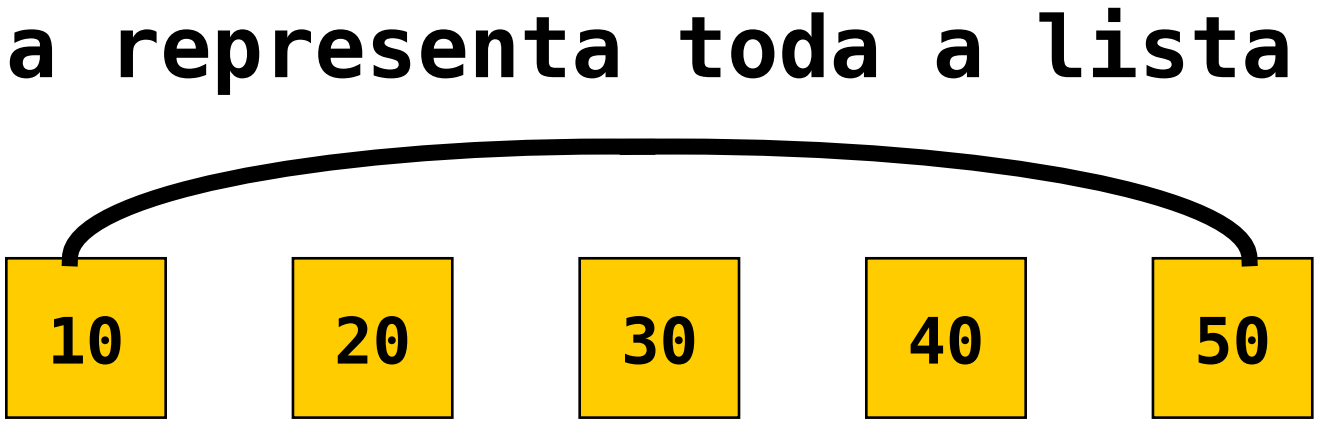
- A nova lista ainda possui quatro elementos, sendo que o último é uma lista. **`[1, 2, 3]`**

8. Pertinência a uma lista

- `names = ['Ann', 'Bob', 'Carol', 'end']`
- `>>> 'Ann' in names`
  `True`
- `>>> 'Lucy' in names`
  `False`
- `>>> 'Alice' not in names`
  `True`
  - *Os mesmos operadores que vimos para strings*

Exemplo

- a=[10, 20, 30, 40, 50]



9. Concatenação de listas

```
>>> names = ['Alice'] + ['Bob']
>>> names
 ['Alice', 'Bob']
>>> names =['Carol']
>>> names = names + 'Dean'
…
TypeError: can only concatenate
 list (not "str") to list
```

9. Concatenação de listas

```
>>> mylist = ['Ann']
>>> mylist*3
['Ann', 'Ann', 'Ann']
>>> newlist = [0]*8
>>> newlist
[0, 0, 0, 0, 0, 0, 0, 0]
>>> [[0]*2]*3
[[0, 0], [0, 0], [0, 0]]
```

10. Fatias (*lists slices*)

```
>>> names = ['Ann', 'Bob',
 'Carol', 'end']
>>> names[0:1]
 ['Ann']
```

- ***Observações:***
  - A ***list slice*** is ***always a list***
  - **`names[0:1]`** inicia com **`names[0]`** mas para antes de **`names[1]`**

10. Fatias (*lists slices*)

```
>>> names[0:2]
['Ann', 'Bob']
```

- Inclui **names[0]** e **names[1]**

```
>>> names[0:]
['Ann', 'Bob', 'Carol', 'end']
```

- Equivale à lista inteira

```
>>> names[1:]
['Bob', 'Carol', 'end']
```

10. Fatias (*lists slices*)

```
>>> names[-1:]
['end']
```

- Uma ***list slice*** é uma lista

```
>>> names[-1]
'end'
```

TRAP WARNING

- *Not the same thing!*

Vamos verificar

- `names=['Ann', 'Bob', 'Carol', 'end']`
- `>>> names[-1]`
  `'done'`
- `>>> names[-1:]`
  `['done']`
- `>>> names[-1] == names[-1:]`
  `False`

O que é verdadeiro para strings **nem sempre** é verdadeiro para listas

11. Quantidades mutáveis e imutáveis

- Listas em Python são ***mutáveis***
  - *Elas podem ser modificadas no local*

```
names = ['Ann', 'Bob', 'Carol', 'end']
>>> names[-1] = 'Astrid'
 >>> names
 ['Ann', 'Bob', 'Carol', 'Astrid']
```

  - *Pode causar surpresas!*

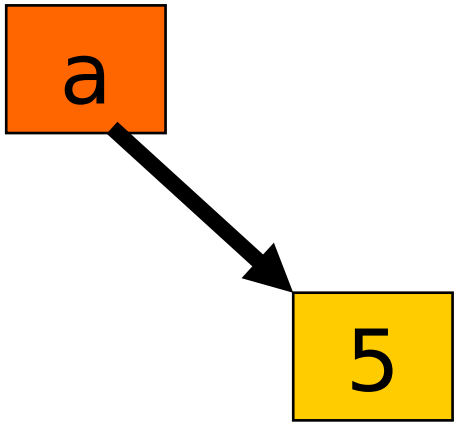
11. Quantidades mutáveis e imutáveis

- Por padrão, as strings em Python **são imutáveis**.
  - ***Cada vez que você as modifica, você cria um novo valor.***

- *Solução computacionalmente mais cara*
  - *Funciona sem **side-effect***

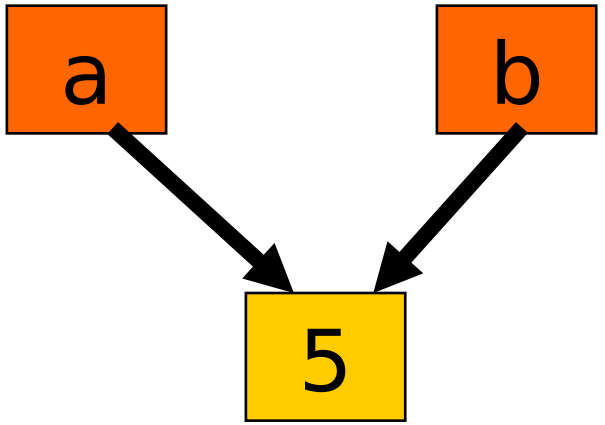
12. Como Python implementa variáveis

- Uma variável em Python contém o endereço de memória de seu valor atual.
- **a = 5**



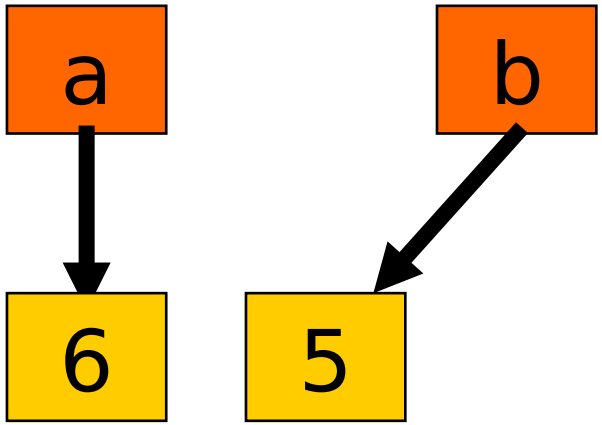
12. Como Python implementa variáveis

- Uma variável em Python contém o endereço de memória de seu valor atual.
- **a = 5**
  b = a



12. Como Python implementa variáveis

- Uma variável em Python contém o endereço de memória de seu valor atual.
- **a = 5**
  b = a
  a = a+1



12. Como Python implementa variáveis

- Cada atribuição gera uma cópia
  - Nós **salvamos** o valor antigo de "a" em "b" antes de modificá-lo.
- O valor anterior de b é salvo
  - **a = 5 # initial value**
  - **b = a # save initial value**
  - **a = a +1 # increment**

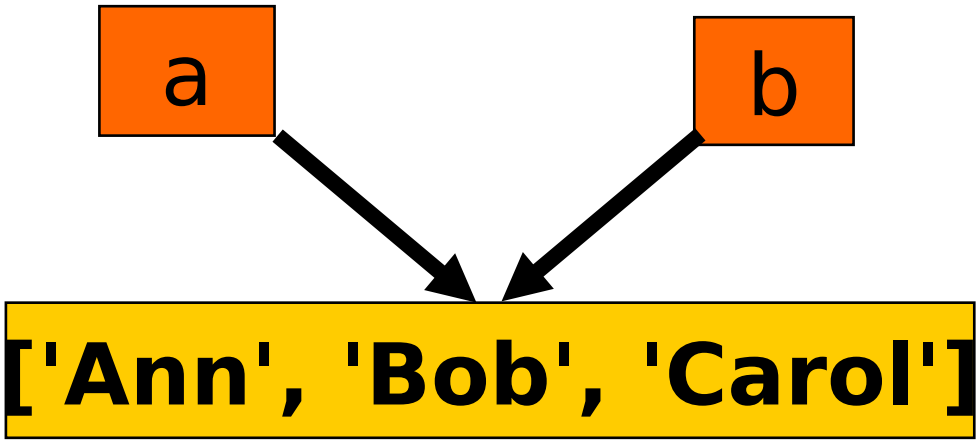
13. Como Python implementa listas

- **>>> a = ['Ann', 'Bob', 'Carol']**
- **>>> b = a**
- **>>> b**
  ['Ann', 'Bob', 'Carol']
- **>>> a[0] = 'Lucy'**
- **>>> a**
  ['Lucy', 'Bob', 'Carol']
- **>>> b**
  ['Lucy', 'Bob', 'Carol']

O valor antigo de "a" **não foi salvo**!

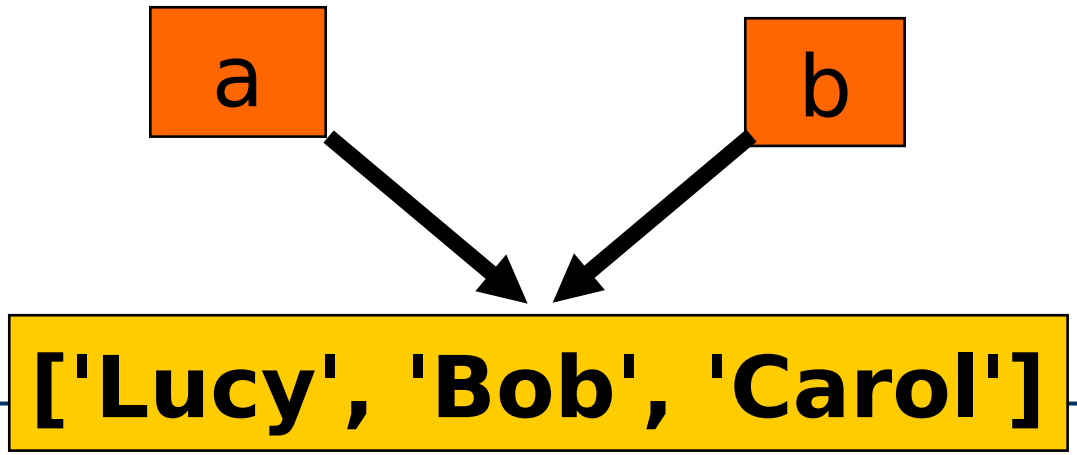
13. Como Python implementa listas

- **>>> a = ['Ann', 'Bob', 'Carol']**
- **>>> b = a**
- **>>> b**
  ['Ann', 'Bob', 'Carol']



13. Como Python implementa listas

- >>> a[0] = 'Lucy'
- >>> a
  ['Lucy', 'Bob', 'Carol']
- >>> b
  ['Lucy', 'Bob', 'Carol']



13. Como Python implementa listas

- Por que isso acontece?
  - Tornar as listas imutáveis teria deixado o Python muito mais lento.
    - *Listas podem ser muito grandes em memória*
  - Conflito entre facilidade de uso e eficiência.
    - Desta vez, a eficiência venceu!
    - Porque a penalidade de eficiência teria sido muito grande.

14. Como copiar uma lista em Python?

- Copiar uma *slice* usando toda a lista '[:]'

```
>>> a = ['Ann', 'Bob', 'Carol']
>>> b = a[:]
>>> b
['Ann', 'Bob', 'Carol']
>>> a[0] = 'Lucy'
>>> a
['Lucy', 'Bob', 'Carol']
>>> b
['Ann', 'Bob', 'Carol']
```

15. Deleções

É possível excluir elementos individuais ou fatias inteiras especificando sua localização.

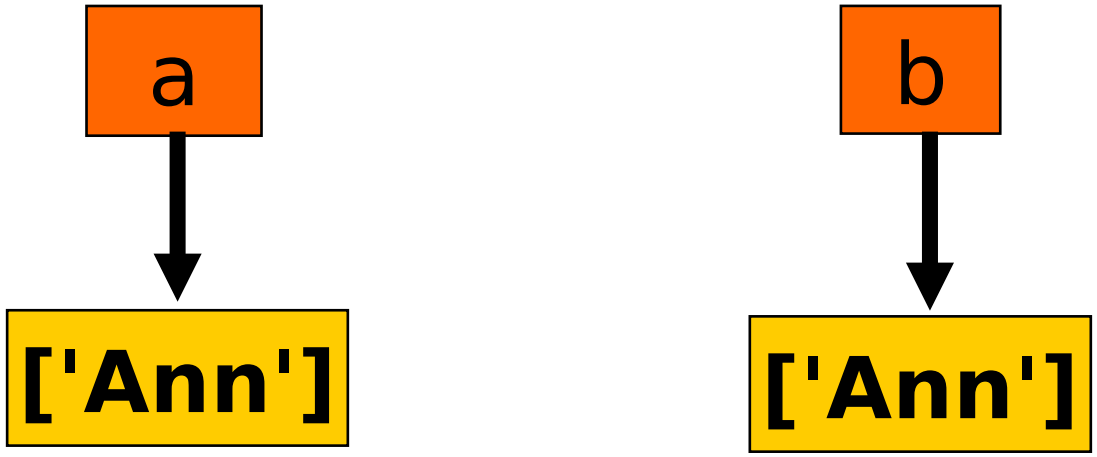
- **names = ['Ann', 'Bob', 'Carol', 'David']**
- **>>> del names[2]**
- **>>> names['Ann', 'Bob', 'David']**
- **>>> del names[0:2]**
- **>>>names**
  ['David']

16. Referências e objetos

- **>>> a = ['Ann']**
- **>>> b = ['Ann']**
- **>>> a == b**
  True
- **>>> a is b**
  False
  - a e b apontam/referenciam para **objetos diferentes!**

16. Referências e objetos

Sem compartilhamento de memória:

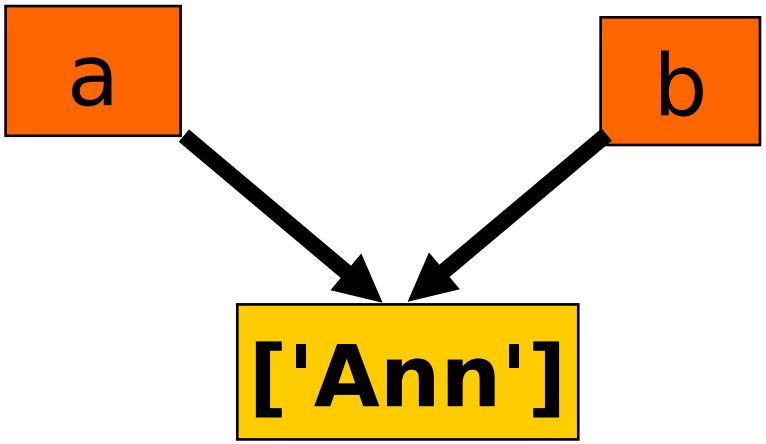


16. Referências e objetos

- **>>> a = ['Ann']**
- **>>> b = a**
- **>>> a == b**
  True
- **>>> a is b**
  True
  - a e b agora apontam para o **mesmo objeto!**

16. Referências e objetos

O mesmo objeto é compartilhado!



17. Aliasing e Cloning

- Quando fazemos:
  - **>>> a = ['Ann']**
  - **>>> b = a**

Não fazemos uma cópia de "**a**", apenas atribuímos um nome adicional de variável ao objeto apontado por "**a**".

- **b** é apenas um *alias* para **a**.

17. Aliasing e Cloning

- Para fazer uma cópia verdadeira de "a", devemos fazer o seguinte:
  - **>>> a = ['Ann']**
  - **>>> b = a[:]**
- **b** é um clone de **a**

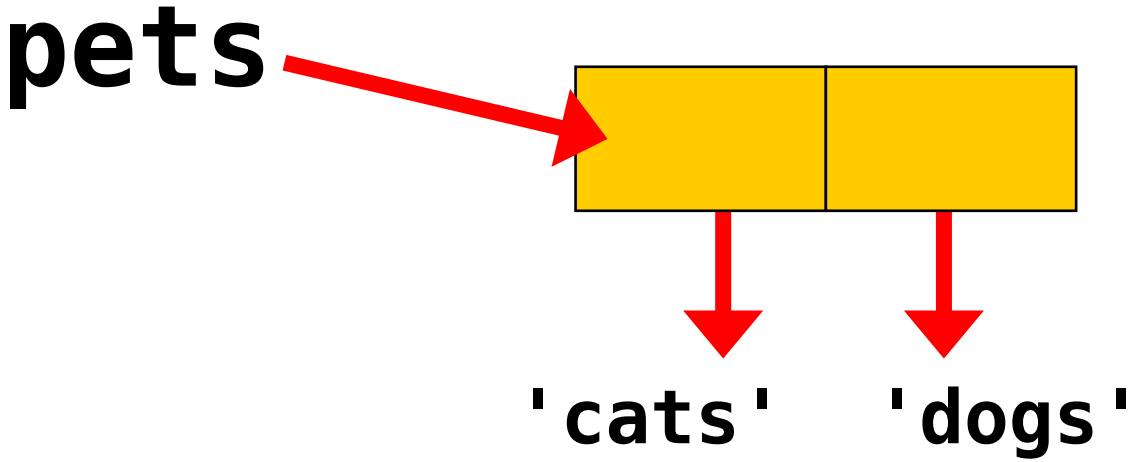
17. Aliasing e Cloning

Um comportamento estranho:

- `>>> pets  = ['cats', 'dogs']`
- `>>> oddlist =[pets]*2`
- `>>> oddlist`
  `[['cats', 'dogs'], ['cats', 'dogs']]`
- `>>> pets[0] = 'snake'`
- `>>> pets`
  `['snake', 'dogs']`
- `>>> oddlist`
  `[['snake', 'dogs'], ['snake', 'dogs']]`

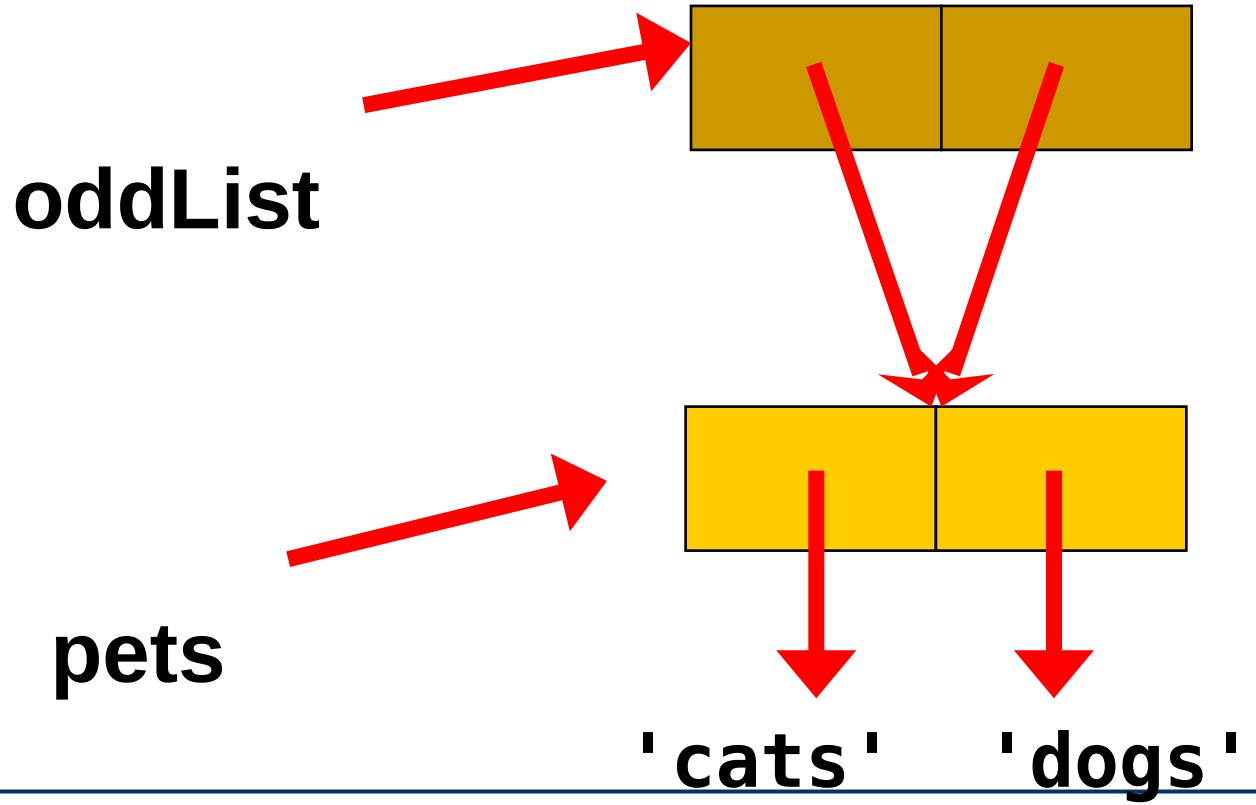
17. Aliasing e Cloning

O que ocorre?



17. Aliasing e Cloning

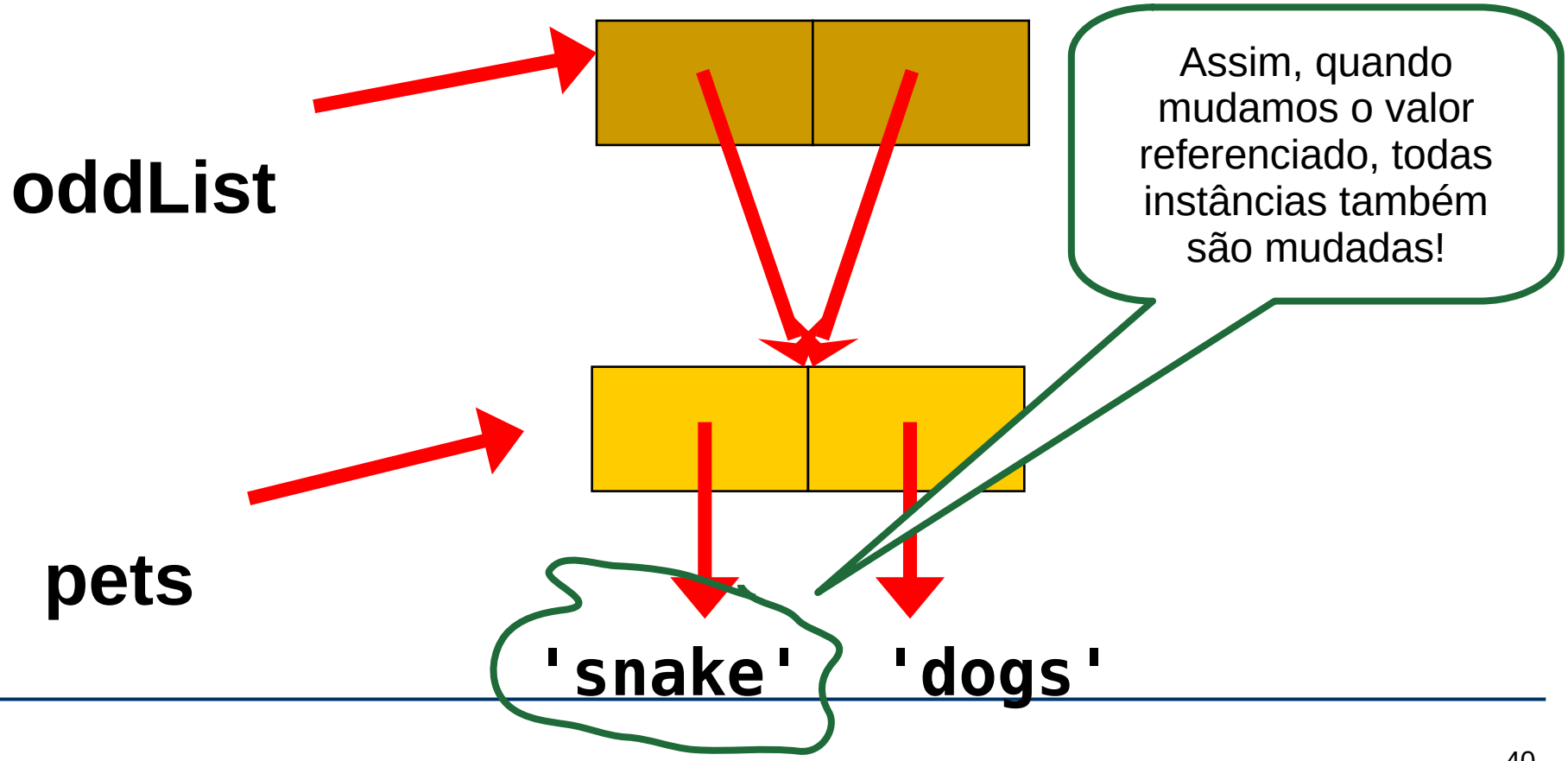
O que ocorre?



Nada é copiado, **apenas referências**!

17. Aliasing e Cloning

O que ocorre?



18. Editando uma lista

```
>>> mylist = [11, 12, 13, 'done']
>>> mylist[-1] = 'finished'
>>> mylist
[11, 12, 13, 'finished']
>>> mylist[0:4] = ['XI', 'XII',
'XIII']
>>> mylist
['XI', 'XII', 'XIII', 'finished']
```

18. Editando uma lista

```
>>> mylist = [11, 12, 13, 'done']
>>> mylist[-1] = 'finished'
>>> mylist
[11, 12, 13, 'finished']
>>> mylist[0:4] = ['XI', 'XII',
'XIII']
>>> mylist
['XI', 'XII', 'XIII', 'finished']
```

Podemos alterar os valores das entradas existentes da lista, mas não podemos adicionar ou excluir nenhuma delas.

19. Métodos da classe 'list'

- Quatro grupos de métodos:
  - Adiciona um item a uma lista e especificando onde colocá-lo
  - Extrai um item de uma lista
  - Modifica a ordem de uma lista
  - Localiza e remove itens individuais

20. Adiciona um item a uma lista e especificando onde colocá-lo

- Dois métodos:
  - **`append()`**
  - **`insert()`**
- Ambos métodos são procedimentos (não retornam nada)

20. Adiciona um item a uma lista e especificando onde colocá-lo

```
>>> mylist=[11,12,13,'finished']
>>> mylist.append('Not yet!')
>>> mylist
[11, 12, 13,'finished','Not yet!']
```

20. Adiciona um item a uma lista e especificando onde colocá-lo

```
>>> listoflists =
[[14.5,'1306'], [17.5,'6360']]
>>> listoflists.append([16.5, 'TAs'])
>>> listoflists
[[14.5,'1306'],[17.5,'6360'],[16.5,'TAs']]
```

Append significa adicionar no final!

20. Adiciona um item a uma lista e especificando onde colocá-lo

```
>>> mylist = [11, 12, 13,'finished']
>>> mylist.insert(0, 10) #BEFORE mylist[0]
>>> mylist
[10, 11, 12, 13, 'finished']
>>> mylist.insert(4, 14) #BEFORE mylist[4]
>>> mylist
[10, 11, 12, 13, 14,'finished']
```

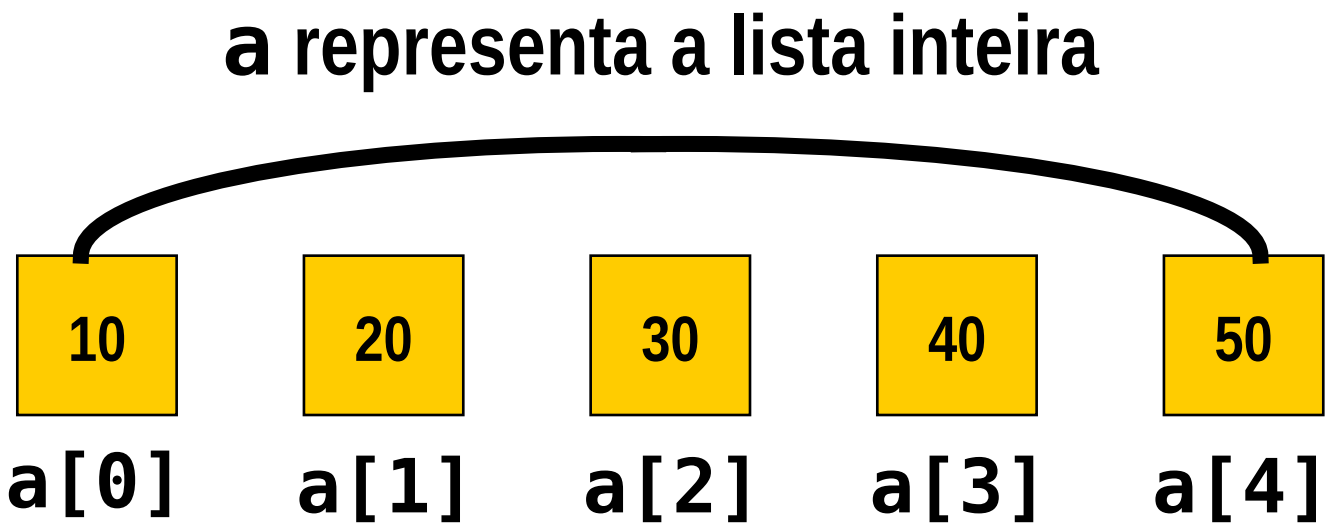
20. Adic
especifi

Sintaxe: **`lista.insert(index, item)`**

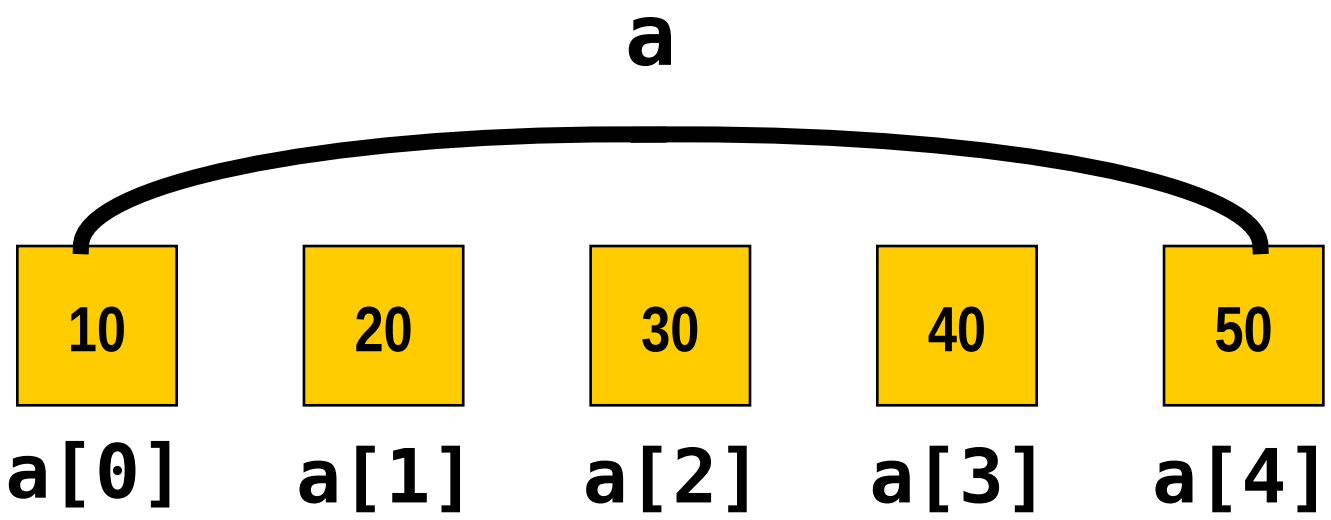
- **`index`** especifica a entrada antes da qual o novo item deve ser inserido
  - **`lista.insert(0, item)`** insere o novo item antes da primeira entrada da lista
  - **`lista.insert(1, item)`** insere o novo item antes do segundo elemento e depois do primeiro

```
>>> myli
>>> myli
>>> mylist
[10, 11, 12,     , 'finished']
>>> mylist.insert(4, 14) #BEFORE mylist[4]
>>> mylist
[10, 11, 12, 13, 14,'finished']
```

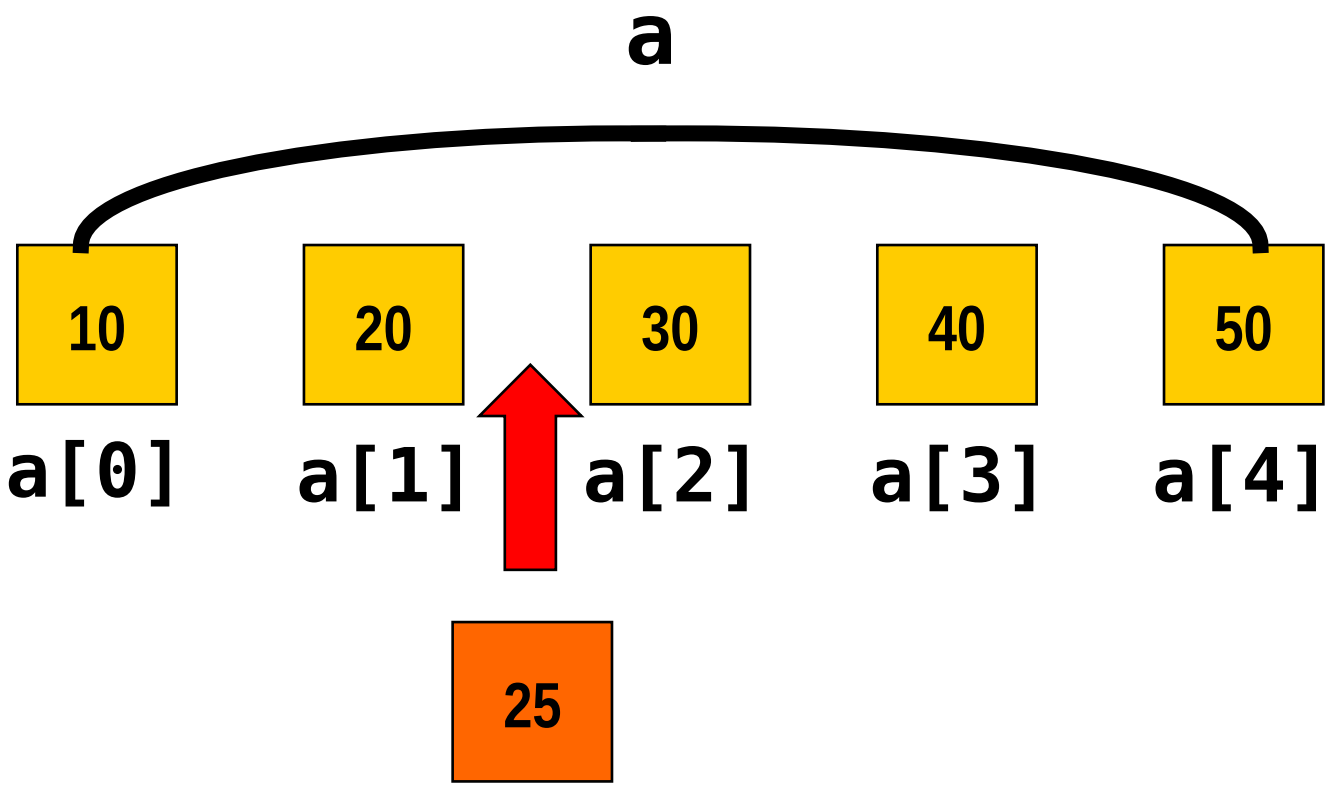
- `a = [10, 20, 30, 40, 50]`



- Onde inserir o valor “25” e manter a lista ordenada?

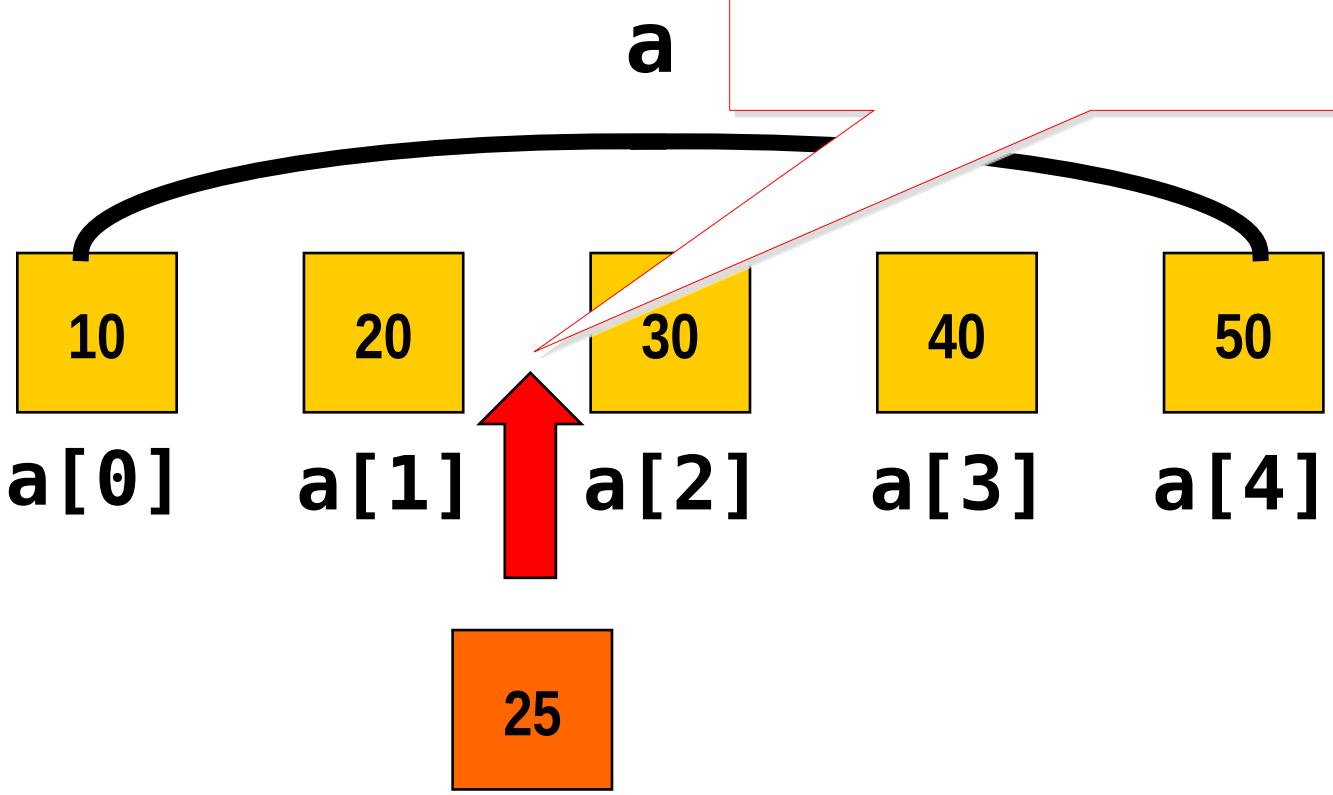


- Onde inserir o valor “25” e manter a lista ordenada?



- Onde inserir o va
  a lista ordenada?

```
>>> a = [10, 20, 30, 40, 50]
>>> a.insert(2,25)
>>> a
[10, 20, 25, 30, 40, 50]
```



Append x Insert

- **`len(list)`**
  - retorna o **comprimento** da lista
    - Mesmo que ***número de elementos da lista***
    - Igual ao último **índice mais um**
- Poderíamos escrever:
  - **`a.insert(len(a), 55)`**
  - Mesma coisa que **`a.append(55)`**


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKURATU
YAQHANYELAY
SUKSAMA
EKHMET
BİYAN
SHUKRIA
TINGKI
MERCI
BOLZÏN
MEHRBANI
GRAZIE
MAAKE
KOMAPSUMNIDA
JUSPAXAR
GOZAIMASHITA
EFCHARISTO
PALDIES

Dúvidas?