Aula 8

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados
  - Listas
  - Listas de Listas

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).

- É uma estrutura de dado recursiva
  - Quando tem zero elementos é representada pela lista vazia: [ ].
  - Quando tem um ou mais elementos, pode ser representada por uma sequência, fechada por colchetes (**[** e **]**), de um ou mais elementos separados por vírgulas.
  - O primeiro elemento da lista está na posição zero.

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).

- É uma estrutura de dado recursiva
  - Quando tem zero elementos é representada pela lista vazia: [ ].
  - Quando tem um ou mais elementos, pode ser representada por uma sequência, fechada por colchetes (**[** e **]**), de um ou mais elementos separados por vírgulas.
  - O primeiro elemento da lista está na posição zero.

- Listas são mutáveis, portanto podem receber novos elementos, substituir elementos existentes ou remover antigos elementos.

Lista

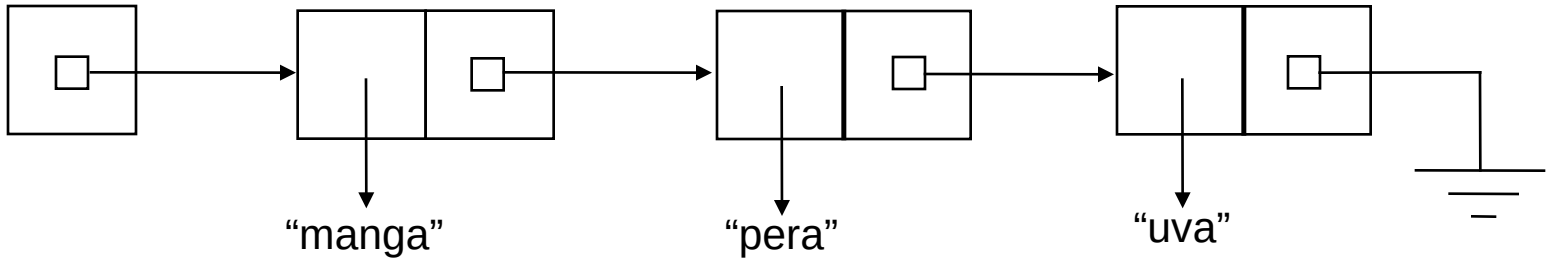
Exemplos:

salada = [ ]

salada

salada = [“manga”, “pera”, “uva”]

salada



Lista (Operações)

- Operações para inclusão de novos elementos:
  - **append(***novoElemento***)**: anexa o *novoElemento* no final da lista;
  - **insert(***pos*, *novoElemento***)**: insere o *novoElemento* na posição *pos* da lista. Caso a lista tenha menos que *pos* elementos, o *novoElemento* é inserido no final da lista.

Lista (Operações)

- Operações para inclusão de novos elementos:
  - **append(***novoElemento***)**: anexa o *novoElemento* no final da lista;
  - **insert(***pos*, *novoElemento***)**: insere o *novoElemento* na posição *pos* da lista. Caso a lista tenha menos que *pos* elementos, o *novoElemento* é inserido no final da lista.
- Exemplos:

```
salada = [“manga”, “pera”, “uva”]

salada.append(“banana”)
# salada = [“manga”, “pera”, “uva”, “banana”]

salada.insert(2, “goiaba”)
# salada = [“manga”, “pera”, “goiaba”, “uva”, “banana”]
```

Exemplo

Criando uma lista de números com quantidade de números aleatórios e um intervalo de valores escolhidos pelo usuário.

```
# Subprogramas
def preencher(listaElems, qtd, min, max):
    from random import randint
    for item in range(qtd):
        listaElems.append(randint(min,max))
    return None
# Programa Principal
qtdNumeros = int(input("A Lista deve ter quantos valores?: "))
minimo = int(input("Menor valor da faixa: "))
maximo = int(input("Maior valor da faixa: "))
numeros = [ ]
preencher(numeros, qtdNumeros, minimo, maximo)
print(numeros)
```

Lista (Operações)

- Operações para remoção de elementos:
  - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
    - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.

Lista (Operações)

- Operações para remoção de elementos:
  - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
    - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.
  - **pop(*pos*)**: retorna e remove o item na posição *pos* da lista.
    - Lança uma exceção “IndexError: pop index out of range”, se aplicado a uma posição inexistente.

Lista (Operações)

- Operações para remoção de elementos:
  - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
    - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.
  - **pop(*pos*)**: retorna e remove o item na posição *pos* da lista.
    - Lança uma exceção “IndexError: pop index out of range”, se aplicado a uma posição inexistente.
  - **remove(*x*)**: remove a primeira ocorrência do item *x*, da esquerda para a direita.
    - Lança uma exceção “ValueError” se x não for encontrado.

Lista (Operações)

- Outras operações úteis sobre listas:
  - *lista*[*pos*]: retorna o elemento da *lista* na posição *pos*.
  - **len(***lista***)**: retorna o comprimento da *lista*.
  - *lista*.**count(***elemento***)**: retorna quantas vezes o *elemento* ocorre na *lista*.
  - *lista*.**sort()**: ordena o conteúdo da *lista*, se os elementos forem todos do mesmo tipo.
    - Caso contrário levanta uma exceção “TypeError: unorderable types”.

Exemplo

Crie uma lista com 100 números aleatórios no intervalo 0 a 40. Remova da lista todos os valores duplicados e mostre seu conteúdo.

```
# Subprogramas
def preencher(listaElems, qtd, min, max):
    ...
def removerDuplicatas(elems):
    indice = 0
    while indice<len(elems):
        if elems.count(elems[indice])==1:
            indice += 1
        else:
            elems.remove(elems[indice])
    return None
# Programa Principal
numeros = [ ]
preencher(numeros, 100, 0, 40)
print(numeros)
removerDuplicatas(numeros)
print(numeros)
```

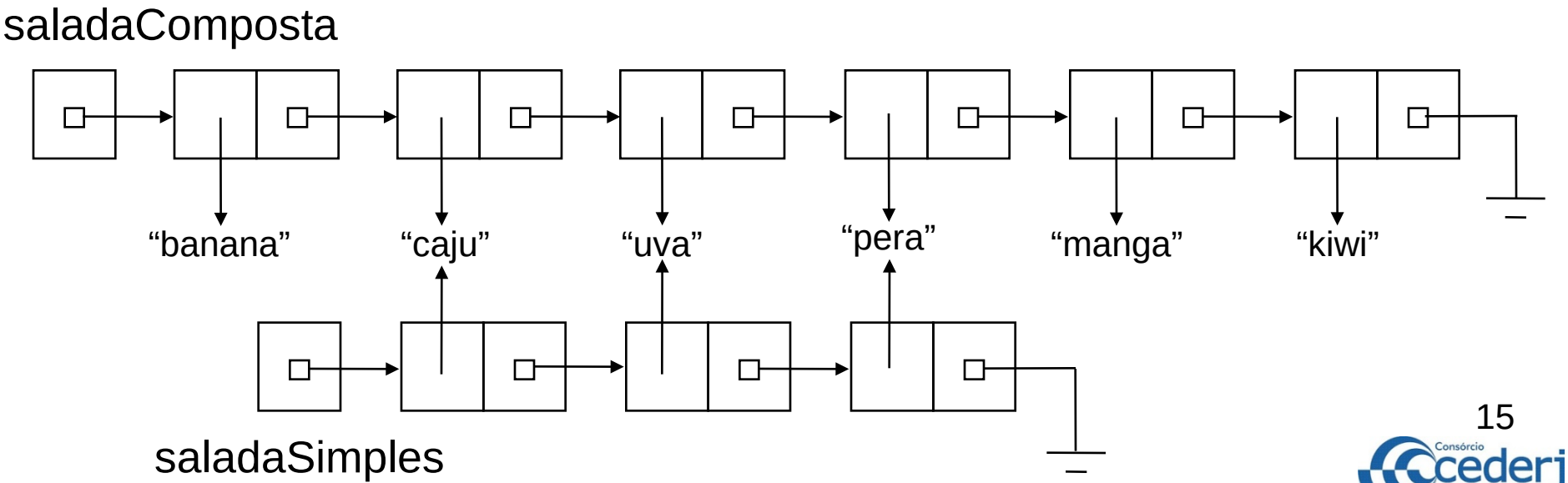
Lista (Operações)

- Fatiamento de Listas:
  - antiga[*posInicio* : *posAposFim*]: retorna um nova lista composta de referências para os elementos existentes, iniciando-se por elemento na posição *posInicio* e finalizando por elemento na posição anterior ao *posAposFim*.

Lista (Operações)

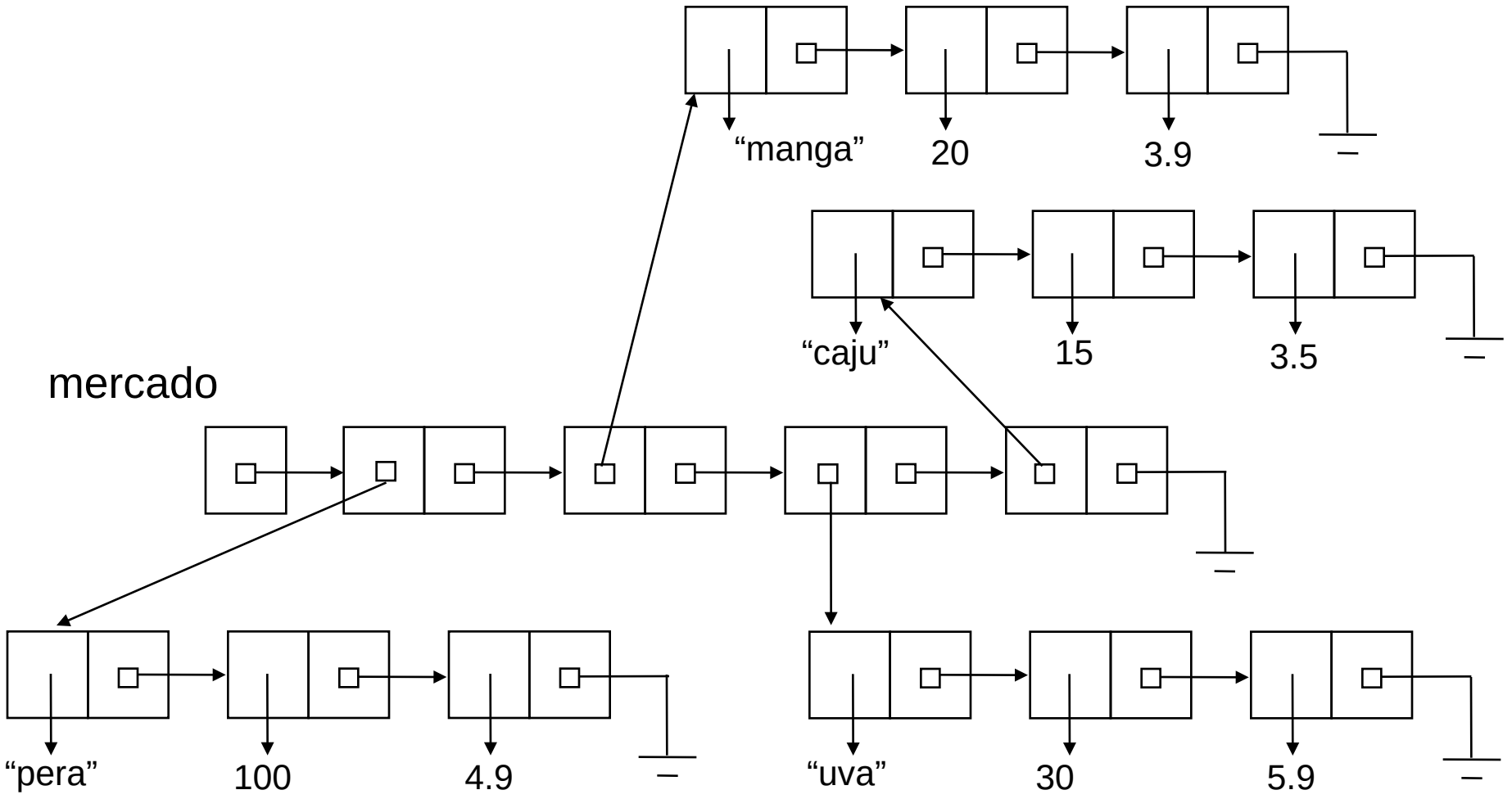
- Fatiamento de Listas:
  - antiga[*posInicio* : *posAposFim*]: retorna um nova lista composta de referências para os elementos existentes, iniciando-se por elemento na posição *posInicio* e finalizando por elemento na posição anterior ao *posAposFim*.
- Exemplo:

saladaComposta = [“banana”, “caju”, “uva”, “pera”, “manga”, “kiwi”]
saladaSimples = saladaComposta[1:4] # lista [“caju”, “uva”, “pera”]



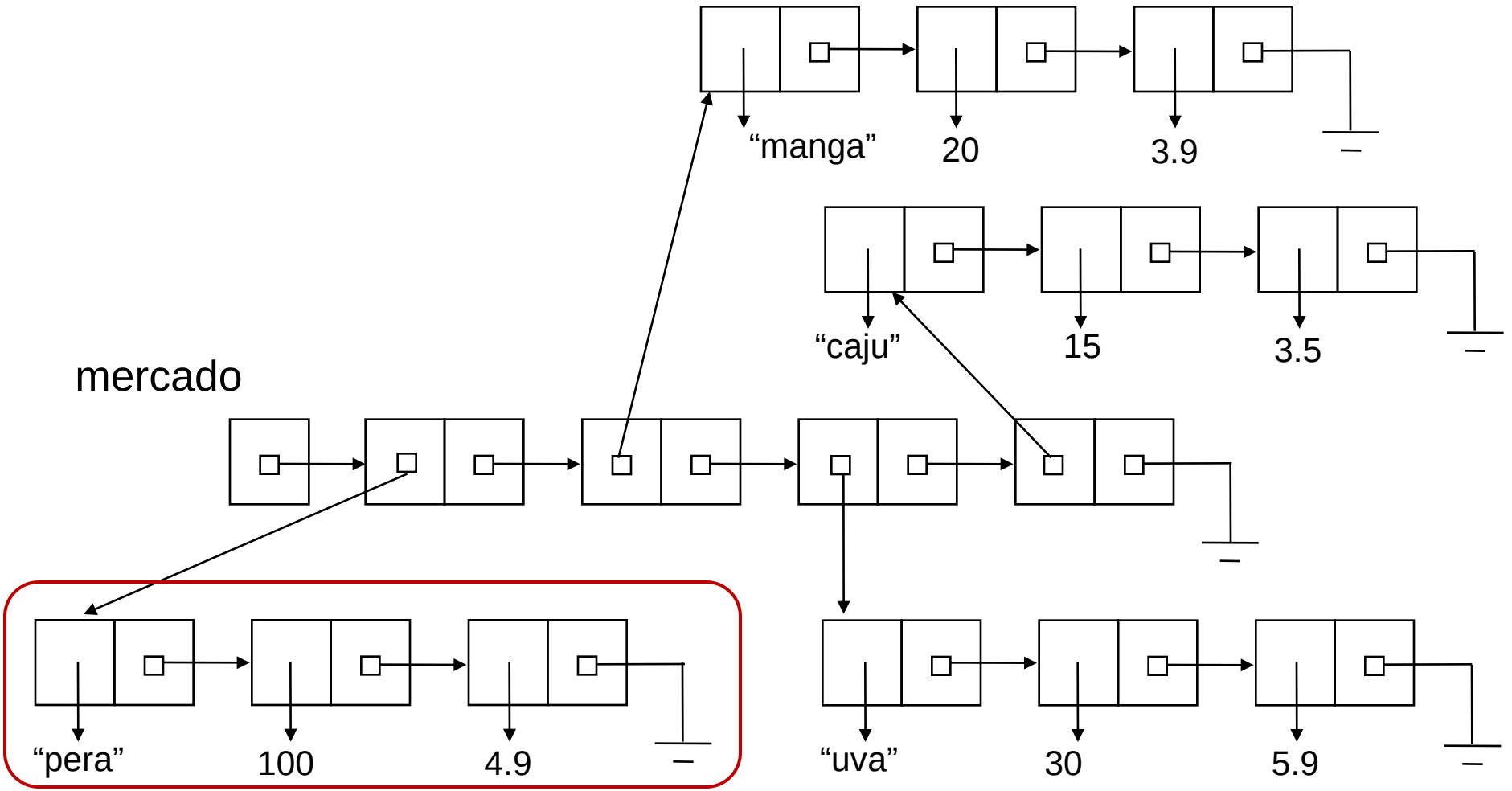
Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



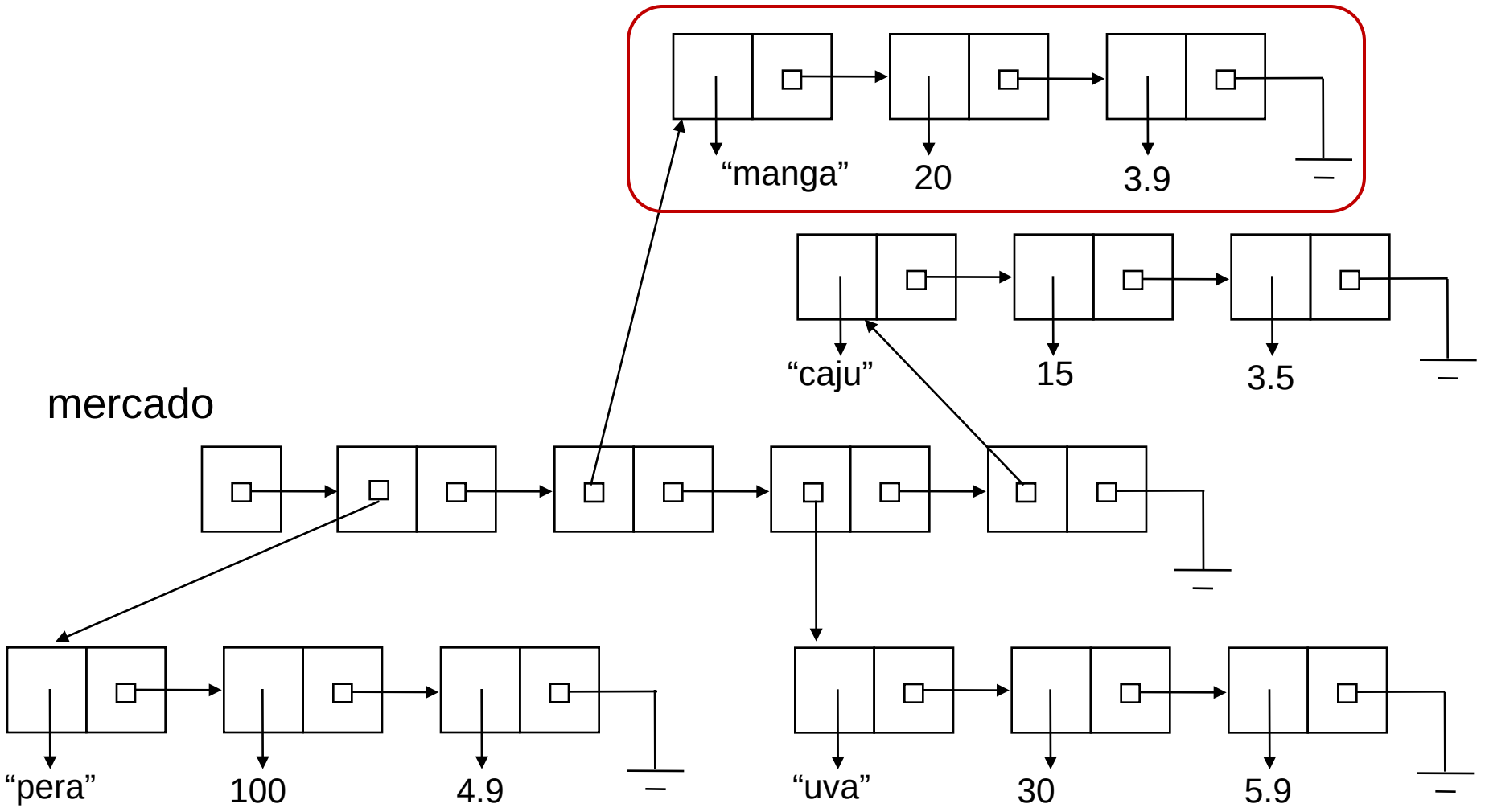
Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



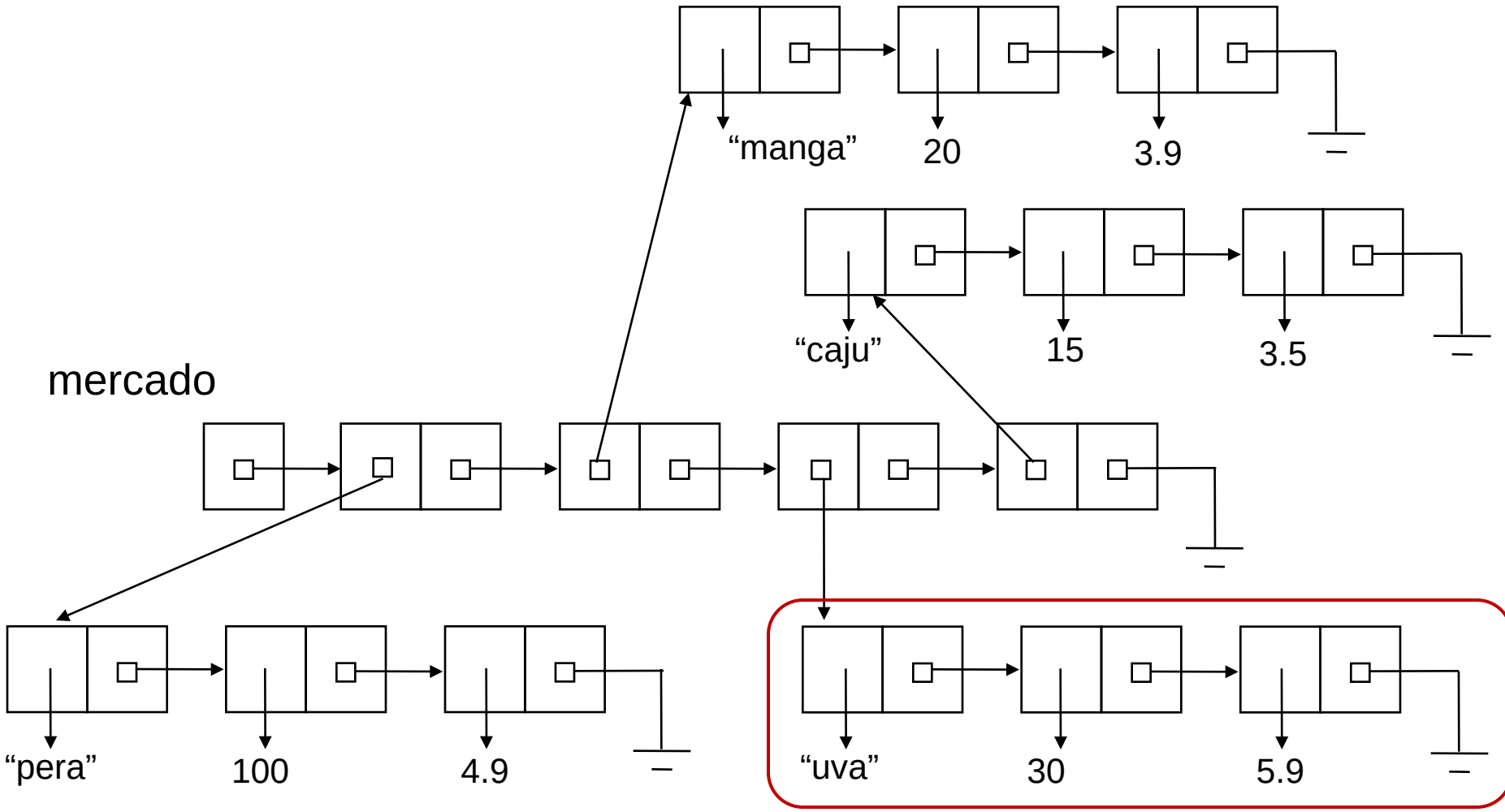
Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



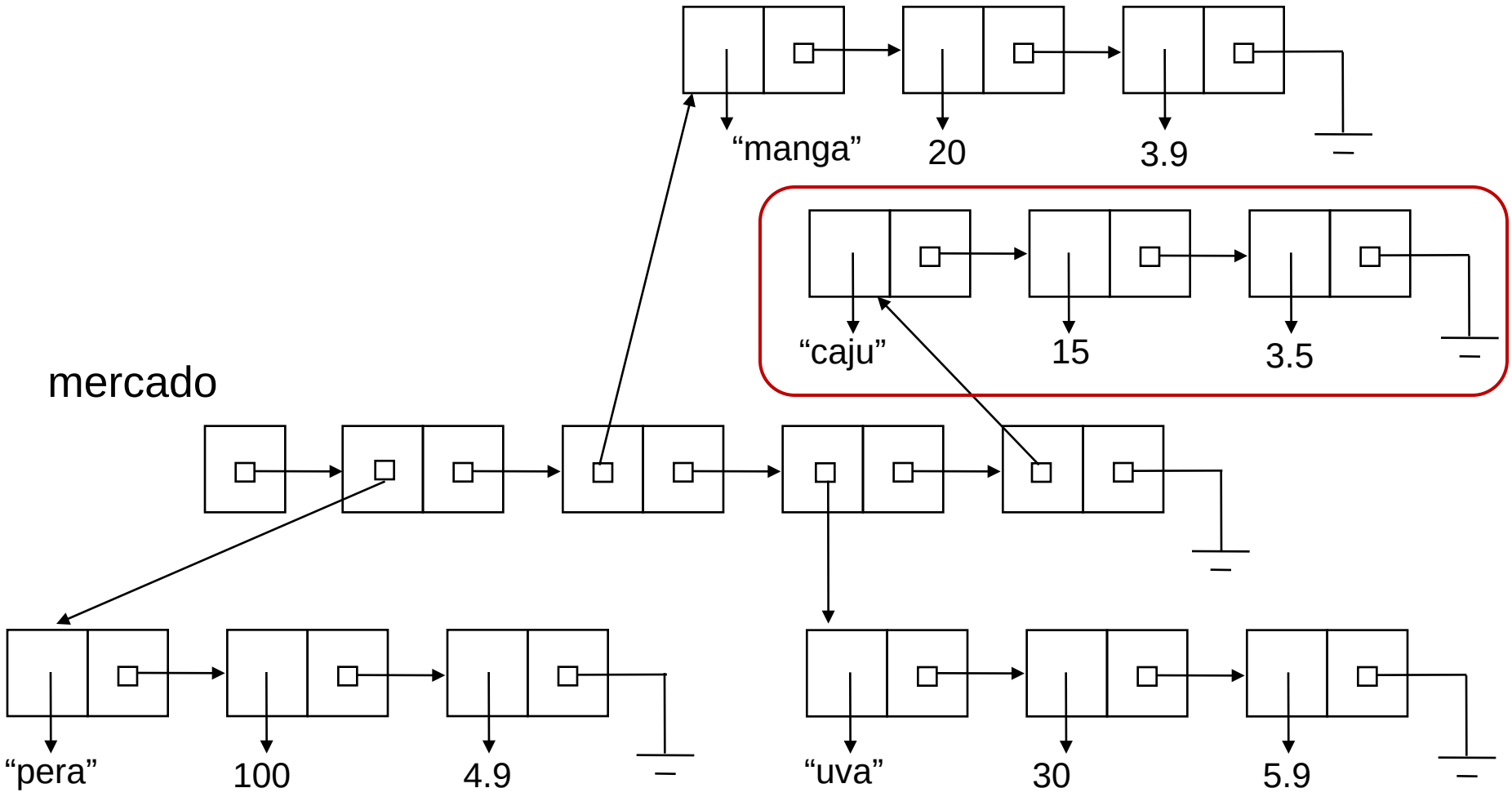
Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



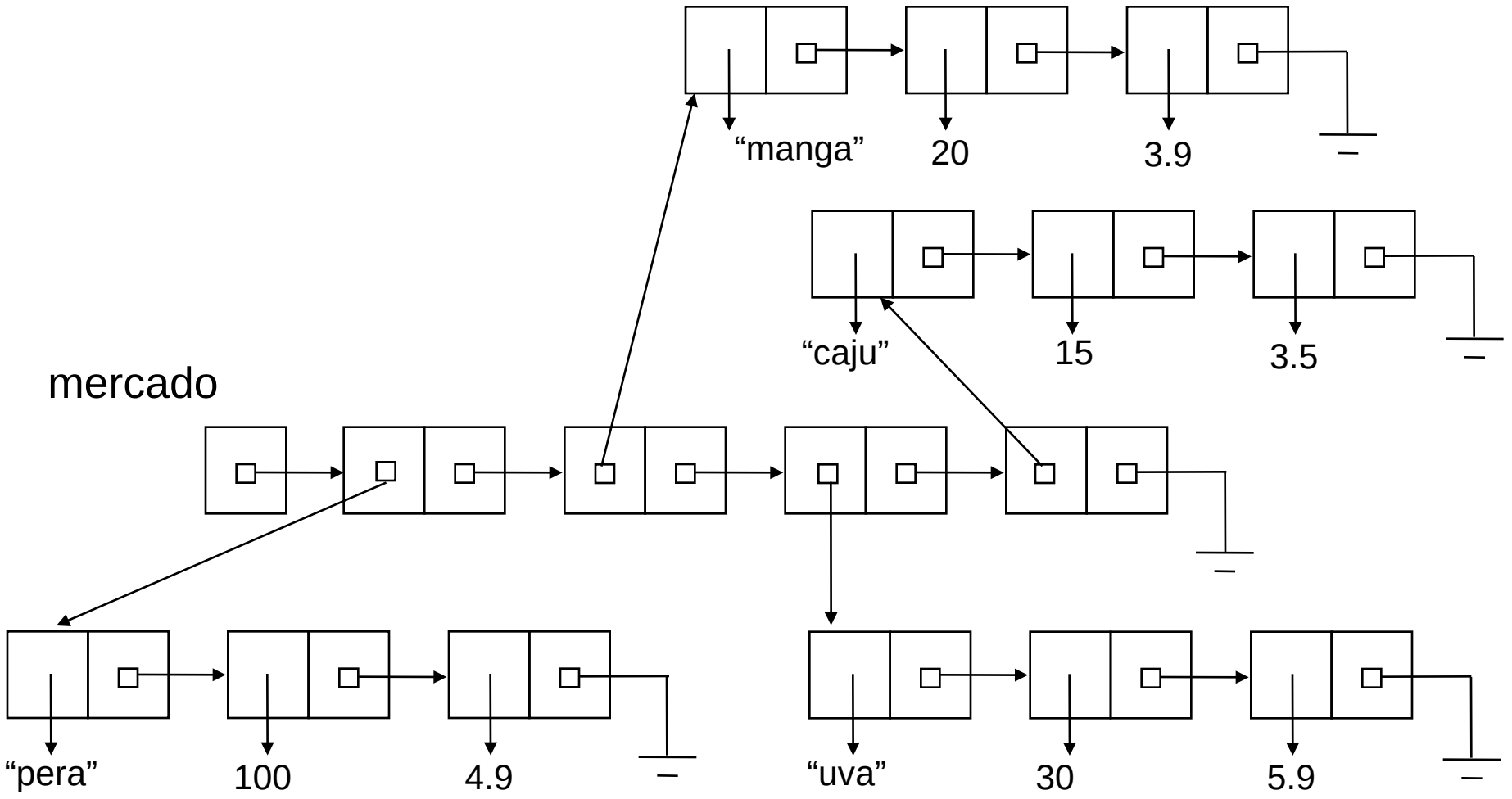
Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



Lista de Listas

mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]



Lista de Listas

```
mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]
print(mercado)

mercado[1][2] *= 0.5                    # manga pela metade do preço
print(mercado)
```

Lista de Listas

```
mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]
print(mercado)

mercado[1][2] *= 0.5                    # manga pela metade do preço
print(mercado)

mercado[3][1] -= 10                     # caju com dez quilos a menos
print(mercado)
```

Lista de Listas

```
mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]
print(mercado)

mercado[1][2] *= 0.5                    # manga pela metade do preço
print(mercado)

mercado[3][1] -= 10                     # caju com dez quilos a menos
print(mercado)

mercado.remove([“uva”,30,5.9])          # o produto uva é removido do mercado
print(mercado)
```

Lista de Listas

```
mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]
print(mercado)

mercado[1][2] *= 0.5                    # manga pela metade do preço
print(mercado)

mercado[3][1] -= 10                     # caju com dez quilos a menos
print(mercado)

mercado.remove([“uva”,30,5.9])          # o produto uva é removido do mercado
print(mercado)

mercado.insert(1, [“kiwi”, 200, 1.99])  # o produto kiwi foi é inserida
print(mercado)
```

Lista de Listas

```
mercado = [[“pera”, 100, 4.9], [“manga”, 20, 3.9], [“uva”, 30, 5.9], [“caju”, 15, 3.5]]
print(mercado)

mercado[1][2] *= 0.5                    # manga pela metade do preço
print(mercado)

mercado[3][1] -= 10                     # caju com dez quilos a menos
print(mercado)

mercado.remove([“uva”,30,5.9])          # o produto uva é removido do mercado
print(mercado)

mercado.insert(1, [“kiwi”, 200, 1.99])  # o produto kiwi foi é inserida
print(mercado)

# mercado = [[“pera”, 100, 4.9], [“kiwi”, 200, 1.99], [“manga”, 20, 1.95], [“caju”, 5, 3.5]]
```

Lista (*Lisp-Like*)

- No sentido de exercitar a elaboração de subprogramas recursivos, apresentamos as três operações primitivas do paradigma de programação funcional da linguagem Lisp (*List Processing*):
  - car(*dados*): é a operação seletora que retorna o primeiro elemento de uma lista dados;
  - cdr(*dados*): é a operação seletora que retorna o resto da lista dados, isto é, retorna uma lista com todos os elementos da lista dados, exceto pelo primeiro;
  - cons(*item*, *dados*): é a operação construtora que retorna uma lista que contém o item como primeiro elemento, seguido pela lista dados.

Lista (*Lisp-Like*)

- No sentido de exercitar a elaboração de subprogramas recursivos, apresentamos as três operações primitivas do paradigma de programação funcional da linguagem Lisp (*List Processing*):
  - car(*dados*): é a operação seletora que retorna o primeiro elemento de uma lista dados;
  - cdr(*dados*): é a operação seletora que retorna o resto da lista dados, isto é, retorna uma lista com todos os elementos da lista dados, exceto pelo primeiro;
  - cons(*item*, *dados*): é a operação construtora que retorna uma lista que contém o item como primeiro elemento, seguido pela lista dados.

- Implementação em Python das operações do Lisp:

```
def car(dados):
    return dados[0]
def cdr(dados):
    return dados[1:len(dados)]
def cons(item, dados):
    return [item]+ dados
```

Processamento Recursivo de Listas (*Lisp-Like*)

- Exemplo:
  - Utilizando as operações seletoras **car** e **cdr**, faça uma função recursiva que some o conteúdo de uma lista de números recebida como parâmetro.

Processamento Recursivo de Listas (*Lisp-Like*)

- Exemplo:
  - Utilizando as operações seletoras **car** e **cdr**, faça uma função recursiva que some o conteúdo de uma lista de números recebida como parâmetro.

- Implementação:

```
def soma(dados):
    if dados == [ ]:
        return 0
    else:
        return car(dados) + soma(cdr(dados))
```

Lista de Listas (*Lisp-Like*)

- Os elementos de uma lista podem ser listas.
  - Portanto, precisamos saber o estado de cada elemento da lista.
  - Para isto, as operações abaixo identificam se um item é uma lista ou se é átomo, isto é, não é uma lista, respectivamente:

```
def ehLista(item):
    return isinstance(item, list)

def ehAtomo(item):
    return not ehLista(item)
```

Lista de Listas (*Lisp-Like*)

- Os elementos de uma lista podem ser listas.
  - Portanto, precisamos saber o estado de cada elemento da lista.
  - Para isto, as operações abaixo identificam se um item é uma lista ou se é átomo, isto é, não é uma lista, respectivamente:

```
def ehLista(item):
    return isinstance(item, list)

def ehAtomo(item):
    return not ehLista(item)
```

- Com estes estados, podemos processar recursivamente uma lista de lista de números e encontrar o valor da soma de todos os valores numéricos:

```
def soma(dados):
    if dados == [ ]:
        return 0
    else:
        if ehAtomo(car(dados)):
            return car(dados) + soma(cdr(dados))
        else:
            return soma(car(dados)) + soma(cdr(dados))
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:

Aula 8

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados
  - Listas
  - Listas de Listas