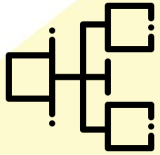
AULA III

22/08/19

FEA.dev

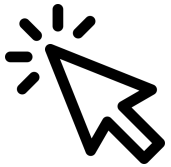
Sumário



Estrutura de Dados

Listas

Índices



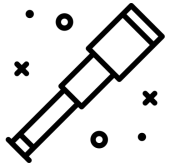
Como acessar valores

.append()

Principais Métodos e Funções com Listas

Funções

Parâmetros de uma função



Escopo de uma função

Estrutura de Dados

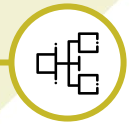
[] Listas

() Tuplas

{:} Dicionários

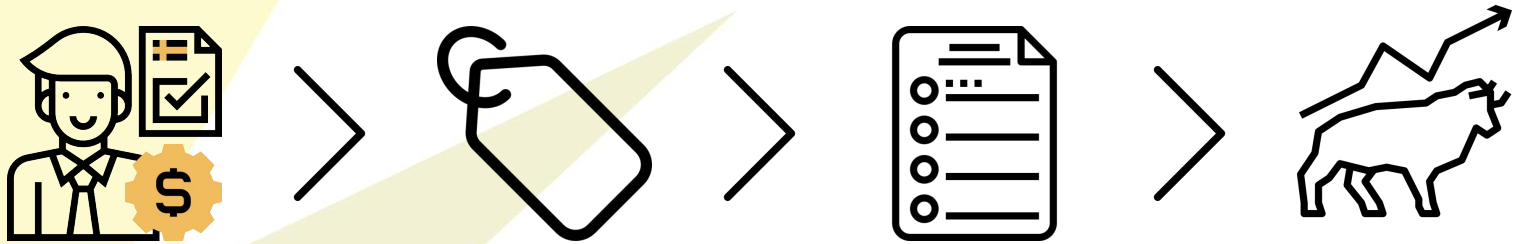
{,} Sets

“Objeto que contém outros objetos”




Listas

“Imagine que você seja um operador da bolsa de valores que esteja acompanhando a tendência de crescimento do valor de uma ação”



Valores dos 4 primeiros meses:

1. *R$ 100,00*
2. *R$ 110,00*
3. *R$ 125,00*
4. *R$ 150,50*

Listas

Valores dos 4 primeiros meses:

1. *R$ 100,00*
2. *R$ 110,00*
3. *R$ 125,00*
4. *R$ 150,50*

```
[1]  # O aluno atribuiria o valor do 1º mês à uma variável x
     x = 100
     x
```

```
100
```

```
[2]  # Agora, faria o mesmo para o 2º mês:
     x = 110
     print(x)
```

```
110
```

Entretanto, note que agora o valor 110 sumiu.

Para resolver isso o aluno poderia atribuir cada valor à uma variável diferente:

```
[3]  mes_1 = 100
     mes_2 = 110
     mes_3 = 125
     mes_4 = 150.50   # note que virgulas são ecritas com ponto
```

Entretanto, perceba que acessar os dados se torna uma tarefa incoveniente:

```
[4]  # Terei que digitar todas as variáveis
     print(mes_1)
     print(mes_2)
     print(mes_3)
     print(mes_4)
```

```
100
110
125
150.5
```

Listas

Solução:

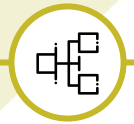
```
[5]  cotacao = [100, 110, 125, 150.5]
     cotacao
```

```
[100, 110, 125, 150.5]
```

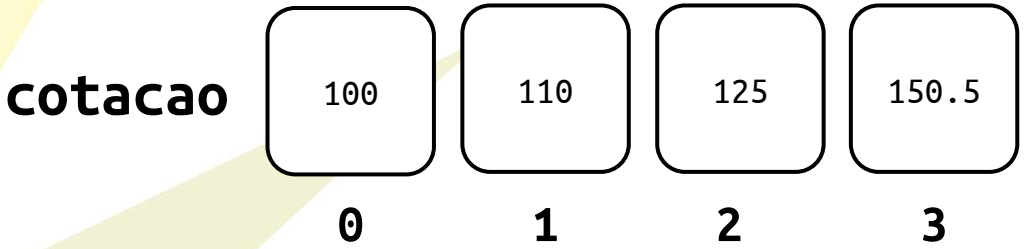
```
[6]  lista_vazia = list()
     lista_vazia = []       # A primeira e a segunda linha fazem a mesma coisa
     lista_vazia
```

```
[]
```

Mas como acessar os valores?




Listas: Índices



Listas: Como acessar valores

```
[10] cotacao

[100, 110, 125, 150.5]
```

```
[7] cotacao[0]

100
```

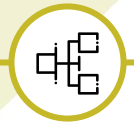
```
[8] cotacao[1]

110
```

```
[9] cotacao[2]

125
```

```
[13] cotacao[0:3]

[100, 110, 125]
```

```
[14] cotacao[:2]

[100, 110]
```

```
[15] cotacao[1:] # vai d

[110, 125, 150.5]
```




Listas: Atribuir valores

"Acessar e igualar"

```
[17]  cotacao[1] = 0
      cotacao
```

```
[100, 0, 125, 150.5]
```

```
[18]  cotacao[2:] = [0, 1]
      cotacao
```

```
[100, 0, 0, 1]
```

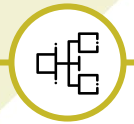
Listas: Principais métodos e funções

```
cotacao [100, 110, 125, 150.5]
```

1) *Método* **`.append()`**

"Adicionar valores ao final da lista"

```
[ ]  cotacao.append(160)
     cotacao

     [100, 110, 125, 150.5, 160]
```



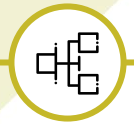

Listas: Principais métodos e funções

```
cotacao [100, 110, 125, 150.5, 160]
```

2) *Método* **`.insert()`**

"Adicionar valor em uma determinada casa"

```
[ ]  cotacao.insert(1, 90)
     cotacao

     [100, 90, 110, 125, 150.5, 160]
```




Listas: Principais métodos e funções

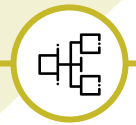
Vamos usar uma lista L:

```
L = [7, 5, 6, 2, 8.3, 4.6, 12, 9]
```

3) *Método* `.sort()`

"Ordena os valores em ordem crescente"

```
[ ]  L=[7, 5, 6, 2, 8.3, 4.6, 12, 9]
     L.sort()
     L

     [2, 4.6, 5, 6, 7, 8.3, 9, 12]
```




Listas: Principais métodos e funções

Ainda com a lista L:

```
L = [7, 5, 6, 2, 8.3, 4.6, 12, 9]
```

4) Método **`.reverse()`**

"Ordena os valores em ordem decrescente"

```
[ ]  L.reverse()
     L

     [12, 9, 8.3, 7, 6, 5, 4.6, 2]
```

Listas: Principais métodos e funções

Retomando a lista cotação:

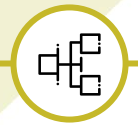
```
cotacao [100, 90, 110, 125, 150.5, 160]
```

5/6) Comando **`del`** *e método* **`.pop()`**

"Exclui um valor especificado da lista, tomando como base o índice"

```
[ ]  del cotacao[5]
     cotacao

     [100, 90, 110, 125, 150.5]
```

```
[ ]  cotacao.pop(1) # Como colocamos 1, a casa 90 será apagada
     cotacao

     [100, 110, 125, 150.5]
```




Listas: Principais métodos e funções

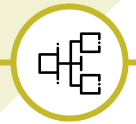
```
cotacao [100, 110, 125]
```

7) *Método* `.remove()`

"Exclui um valor especificado da lista, tomando como base o valor"

```
[ ]  cotacao.remove(100) # Irá apagar a casa de indice 0, já que ela que possui o valor 100
     cotacao
```

```
[110, 125]
```




Listas: Principais métodos e funções

8) *Função* **`range()`**

"Cria uma sequência de números"

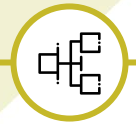
```
[ ]  list(range(0,5))

     [0, 1, 2, 3, 4]
```

```
[ ]  list(range(2,5))

     [2, 3, 4]
```

9) *Função* **`len()`**

"Retorna o tamanho de um objeto"

```
[ ]  lista = [3, 2, 6, 0]
     print(lista)
     print(len(lista))

     [3, 2, 6, 0]
     4
```

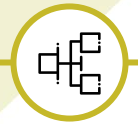



Listas: “Nested” lists

Listas podem conter outras listas? → *Sim!*

Vamos criar uma lista de indivíduos, e cada indivíduo vai possuir um nome e uma idade.

```
pessoas = [ind1, ind2, ind3, ...]
ind → [‘nome’, idade]
pessoas = [[‘Pedro’, 25], [‘Maria’, 18], [‘Carlos’, 20]]
```




Listas: “Nested” lists

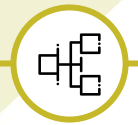
Acessando valores

```
pessoas = [[‘Pedro’, 25], [‘Maria’, 18], [‘Carlos’, 20]]
```

`pessoas[0]` → `[‘Pedro’, 25]`

Para acessar os valores dentro da lista menor basta colocarmos mais colchetes

`pessoas[0][1]` → `[25]`

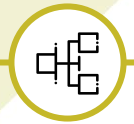



Funções

Algumas usadas até agora:

`print()` `len()` `range()` `list()`

"Compilar operações em uma operação só.
Assim, evitamos escrever o mesmo código várias vezes"




Funções

Voltando ao nosso exemplo

E se quiséssemos formatar os dados da lista?

Proposta

Criar uma função que escreva na tela um separador

Sintaxe

```
def nome_da_funcao(a, b):
    operação
```

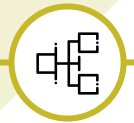
Funções

Sintaxe

```
def nome_da_funcao(a, b):
    operação
```

Cuidado!

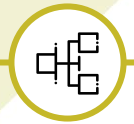
Indentação




Funções

"Cara" do script:

```
def separador():
    print('--Valor--')

separador()
print(cotacao[0])
separador()
print(cotacao[1])
separador()
print(cotacao[2])
separador()
print(cotacao[3])
```




Funções: Parâmetros

Assim como na matemática, funções recebem parâmetros (ou variáveis)

$$z = f(x, y) = x + y$$

```
def separador(mes):
    print('-- Mes {} --'.format(mes))
```

```
[ ]  def separador(mes):
       print('-- Mes {} --'.format(mes))


     separador('1')
     print(cotacao[0])
     separador('2')
     print(cotacao[1])
     separador('3')
     print(cotacao[2])
     separador('4')
     print(cotacao[3])
```

```
-- Mes 1 --
110
-- Mes 2 --
110
-- Mes 3 --
125
-- Mes 4 --
150.5
```

Funções: Parâmetros

Alguns cuidados:

- *separador()* =/= *Separador()*
- *Deixar de inserir parâmetros*
  - *Parâmetros padrões*
- *return* $z = f(x, y) = x + y$

Funções: Escopo

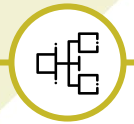
"Variável local e variável global"

```
[ ]  def funcao(b):
       b += 4
       c = 2
       print('B dentro vale {}'.format(b))
       print('C dentro vale {}'.format(c))

     a = 5
     funcao(a)
     print('A fora vale {}'.format(a))
```

```
B dentro vale 9
C dentro vale 2
A fora vale 5
```

```
[ ]  def funcao(b):
       a = 8
       b += 4
       c = 2
       print('A dentro vale {}'.format(a))
       print('B dentro vale {}'.format(b))
       print('C dentro vale {}'.format(c))

     a = 5
     funcao(a)
     print('A fora vale {}'.format(a))
```

```
A dentro vale 8
B dentro vale 9
C dentro vale 2
A fora vale 5
```

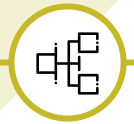


Funções: Escopo

"Operador global"

```
[ ]  def funcao(b):
       global a
       a = 8
       b += 4
       c = 2
       print('A dentro vale {}'.format(a))
       print('B dentro vale {}'.format(b))
       print('C dentro vale {}'.format(c))

     a = 5
     funcao(a)
     print('A fora vale {}'.format(a))
```

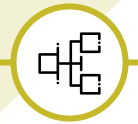
```
A dentro vale 8
B dentro vale 9
C dentro vale 2
A fora vale 8
```




Observações

- *print(f'O valor digitado foi {valor}')*
- *print(f'Valor com duas casas decimais {valor:.2f}')*
- *print('Esse texto \n será quebrado')*
- *# → Comentários*

  '''
- *Comentário longo*
  '''




FEA.dev

OBRIGADX!

/fea.dev

diegolottermann@usp.br
miguel_sallum@usp.br

dlotter e msallum

Av. Prof. Luciano Gualberto, 908 -
Butantã, São Paulo/SP