Fundamentos de Programação

Estruturas de DADOS

Listas

Aula 13

Post It da Aula Passada

O Comando *While*

- Estrutura genérica do comando
- Critério de parada
  - Expressão relacional ou lógica
- Indentação

Outros Recursos

- O comando *break*
  - Para uma estrutura de repetição durante uma iteração
- Estruturas de repetição aninhadas

A Intuição de Listas

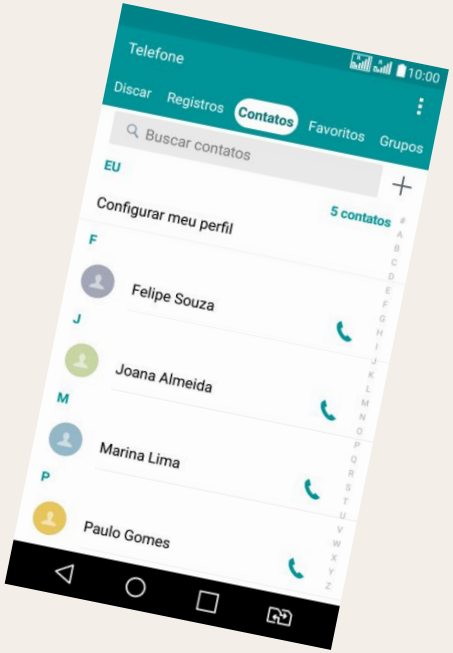
Listas são extremamente comuns no nosso cotidiano!

Fazemos listas com diversas finalidades:

- Lista de tarefas
- Lista de compras
- Lista telefônica

Em geral, listas apresentam um **padrão estrutural bem definido**: uma coleção ordenada de valores, sejam estes valores únicos ou não

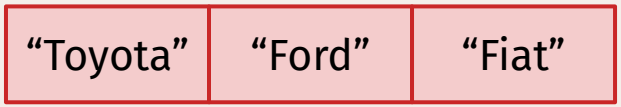
A Intuição de Listas



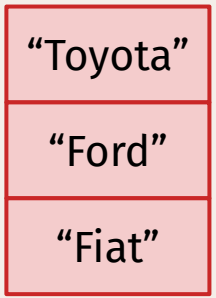
A Intuição de Listas

Porém, na prática, uma lista nada mais é que uma **estrutura de dados abstrata**!

De um ponto de vista representativo podemos apresentar a mesma lista de formas diferentes, por exemplo:



Horizontal



Vertical

A Intuição de Listas

Uma lista, apesar de ser compreendida com uma única estrutura, é **tipicamente formada por múltiplos elementos**

Primeiro elemento: “Toyota”
Segundo elemento: “Ford”
Terceiro elemento: “Fiat”

O Tipo Lista

Muitas linguagens de programação não proveem listas como um tipo de dados nativo. Nesse caso, existem diversas técnicas para a implementação de listas, sejam estas utilizando vetores ou sejam utilizando ponteiros

Porém, para a nossa alegria, Python disponibiliza a estrutura de dados lista como um tipo nativo!

Mais do que disponibilizar, as listas em Python são bastante flexíveis e **podem armazenar elementos de diferentes tipos de dados** em simultâneo

O Tipo Lista

Mas, como utilizar listas em Python?

1. A notação para a definição de uma lista em Python é dada por dois colchetes: **[]**
2. Listas podem ser iniciadas vazias (**[]**), iniciadas com dados de um único tipo (**[1, 2, 10, 20]**) ou ainda iniciadas com dados de diferentes tipos (**["ABC", 10, 5.2, True]**)
3. Listas são naturalmente atribuíveis em variáveis utilizando o operador de atribuição (**var_lista = [1, 5, 10, 15]**)

O Tipo Lista

Faça o teste! Crie as seguintes listas em um documento Python:

var_lista_1 = **[]**
var_lista _2 = **[“abacate”, “abacaxi”, “limão”, “laranja”, “banana”]**
var_lista _3 = **[10.0, 11.0, 11.4, 12.8, 13.5]**

Agora, exiba as listas criadas na tela utilizando a função *print*:

print(var_lista_#)

Trabalhando com Listas

Lembram da função *len*? Aquela que utilizamos para verificar a quantidade de caracteres existentes em um dado do tipo *string* (str)

Essa função também funciona para listas, mas ao invés de retornar a quantidade de caracteres, neste contexto, **a função *len* retorna a quantidade de elementos** presentes na lista provida como argumento

```
print("Tamanho da lista #:", len(var_lista_#))
```

Trabalhando com Listas

Neste ponto, vale a pena já conhecermos também o **comando *in***. Este comando, na verdade, é um operador binário que verifica se o elemento provido a sua esquerda está contido na lista provida à sua direita:

elemento in *lista*

O resultado (retorno) do comando in é sempre um valor lógico: *True*, se o elemento está contido na lista; ou *False*, se o elemento não está contido na lista

print(“abacaxi” in var_lista_#)

Trabalhando com Listas

As listas também **funcionam naturalmente como um espaço de iteração**, sendo utilizáveis como tal na definição de uma estrutura de repetição *for*:

```
for elemento in var_lista_#:
    print(elemento)
```

Também, como fazíamos para as *strings*, **podemos acessar elementos de uma lista através dos seus respectivos índices**, sendo que tais índices variam no intervalo [0, len(*lista*)-1]

Trabalhando com Listas

O acesso a elementos pelos seus índices acontece da mesma forma que em dados do tipo *string*:

lista[*índice*]

Por exemplo:

```
print(var_lista_2[0])
print(var_lista_3[2])
```

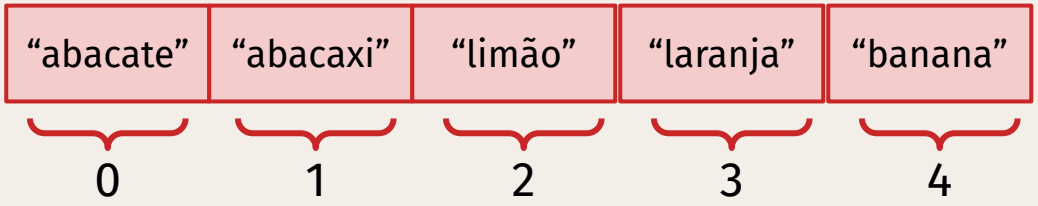
E se excedemos o limite de índices válidos para uma lista?

print(var_lista_1[0])

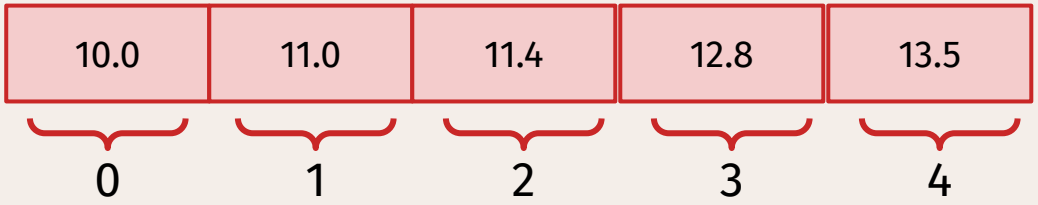
Trabalhando com Listas

var_lista_1 ∅

var_lista_2



var_lista_3



Trabalhando com Listas

Dado que o acesso a elementos mediante índices é disponibilizado, também podemos **utilizar a estrutura de repetição *for* em conjunto com a função *range* para iterar sobre os elementos de uma lista**

```
for i in range(len(var_lista_#)):
    print(“Elemento:”, var_lista_#[i], “  Índice:”, i)
```

Trabalhando com Listas

Até aqui podemos pontuar que:

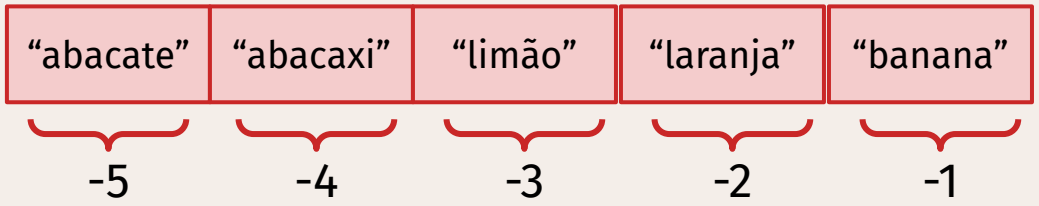
- Uma lista pode ser compreendida, em Python, como uma **coleção de elementos alocados de maneira ordenada**
- Todo o **elemento presente em uma lista tem uma posição** correspondente, essa posição é o seu índice
  - Índices estão contidos no intervalo [0, len(*lista*)-1]
- Listas funcionam naturalmente como **espaços de iteração** em estruturas de repetição *for*, mas também podemos iterar as mesmas através de seus índices (funções *range* + *len*)

Trabalhando com Listas

Uma característica interessante da manipulação de listas através dos índices dos seus elementos é o percorrimento invertido da lista (do último para o primeiro)

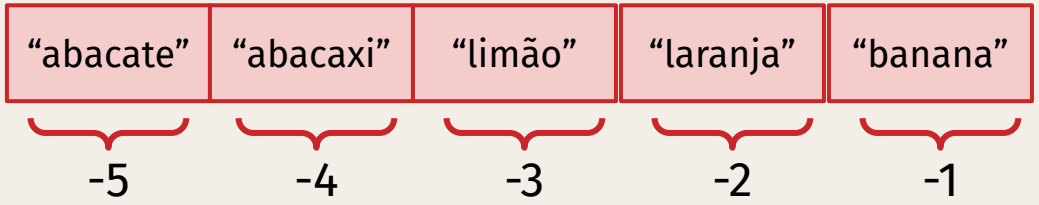
Podemos acessar **o último elemento da lista utilizando o índice -1, o penúltimo utilizando o índice -2, e assim por diante** até que acessamos o primeiro elemento da lista usando o índice -len(lista)

var_lista_2



Trabalhando com Listas

var_lista_2



print(var_lista_2[-1]) print(var_lista_2[-5]) print(var_lista_2[-7])

Também podemos adaptar o laço for para fazer o percorrimento invertido da lista:

```
for i in range(len(var_lista_2)):
    print(var_lista_2[-i - 1])
```

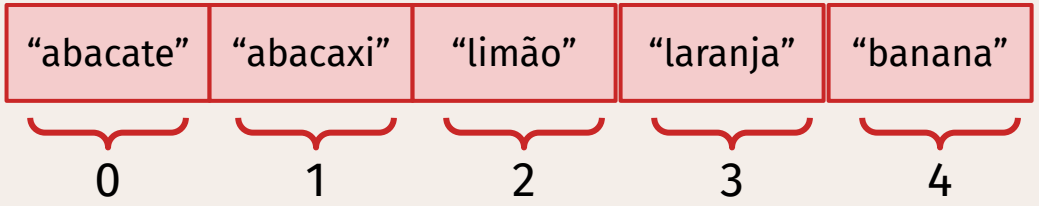
Trabalhando com Listas

Mais uma similaridade com as strings: **podemos utilizar a manipulação *head/tail* no contexto de listas** da exata mesma maneira que utilizávamos para *strings*

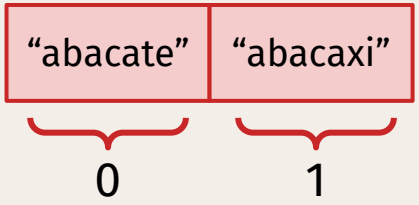
Relembrando: **a manipulação *head/tail* utiliza a notação [n:m]**, onde *n* indica a quantidade de elementos removidos a partir do início da lista e *m* indica a quantidade de elementos inseridos a partir do início da lista, **sendo *m* sempre resolvido antes de *n***

Trabalhando com Listas

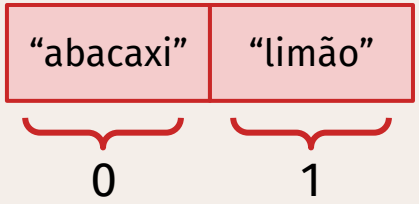
var_lista_2



var_lista_2[:2]



var_lista_2[1:3]



Funções de Lista

Como acontece para outros tipos de dados, listas apresentam diversas funções que podem ser acessadas através da notação ponto. Vamos ver algumas das principais:

***lista*.count(*elemento*)**
Retorna a quantidade de ocorrências de *elemento* em *lista*
print(var_lista_2.count(“abacaxi”))

Funções de Lista

***lista*.index(*elemento*)**
Retorna o índice (posição) do *elemento* na *lista*, o *elemento* deve estar contido na *lista*
print(var_lista_2.index(“abacaxi”))

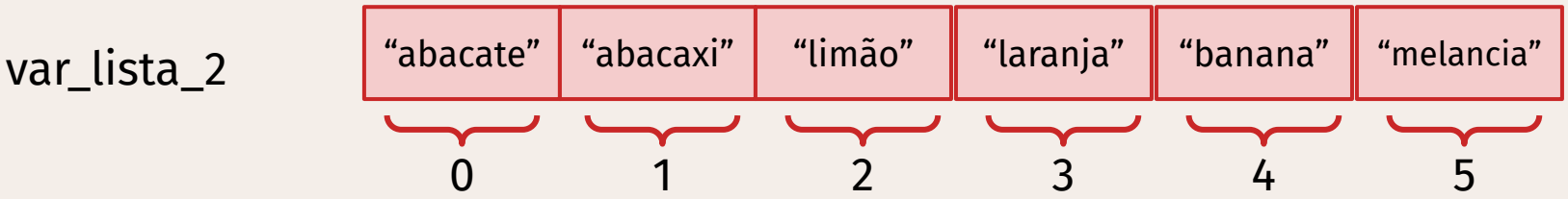
***lista*.sort()**
Ordena os elementos em ordem crescente de valor ou lexicográfica
var_lista_2.sort()
print(var_lista_2)

Funções de Lista

***lista*.append(*elemento*)**

Insere um elemento no final da lista, a lista passa a ter mais um elemento e, consequentemente, mais uma unidade em seu tamanho

```
var_lista_2.append(“melancia”)
print(var_lista_2)
```

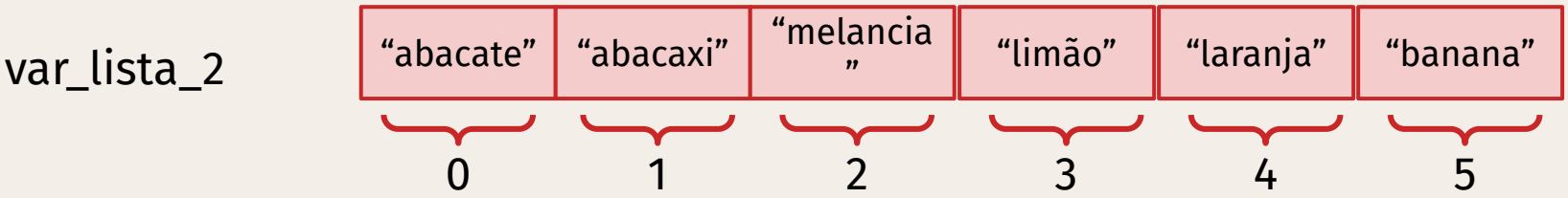


Funções de Lista

***lista*.insert(*índice*, *elemento*)**

Insere um elemento no índice indicado da lista, os elementos posteriores já inseridos na lista são alocados em uma posição posterior. Se o índice indicado superar o tamanho da lista, a função *insert* funciona como a função *append*

```
var_lista_2.insert(2, “melancia”)
print(var_lista_2)
```

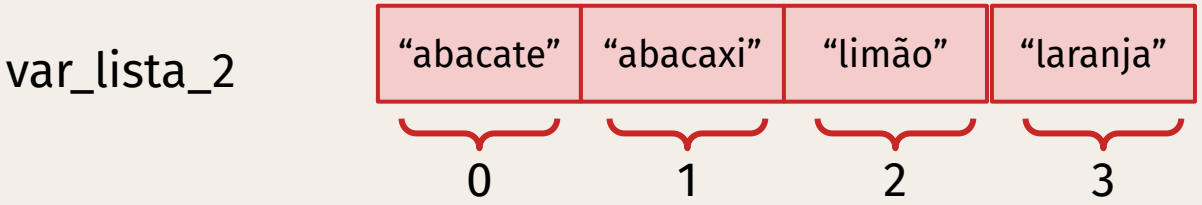


Funções de Lista

***lista*.pop()**

Remove o último elemento da lista e o retorna como resultado da função

```
elem_removido = var_lista_2.pop()
print(var_lista_2)
print(elem_removido)
```

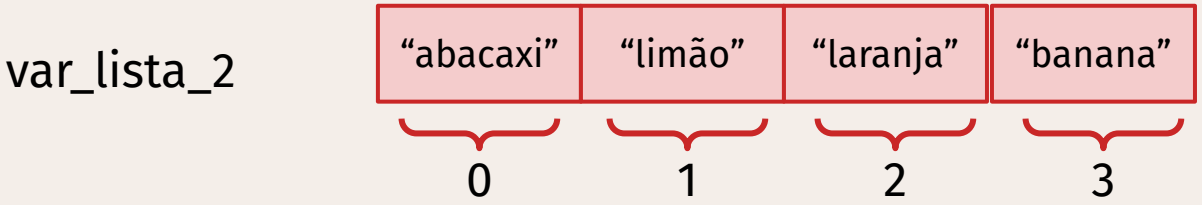


Funções de Lista

***lista*.remove(*elemento*)**

Remove a primeira ocorrência de um elemento da lista, os elementos posteriores já inseridos na lista são alocados em uma posição anterior

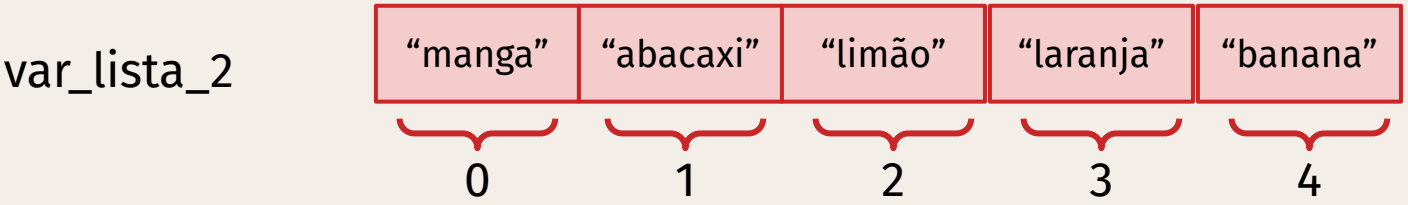
```
var_lista_2.remove(“abacate”)
print(var_lista_2)
```



Atribuição em Listas

Diferente das *strings*, em listas também podemos **manipular diretamente um dado através de seu índice fazendo atribuições diretamente na sua posição correspondente** (assim substituindo o dado lá presente anteriormente)

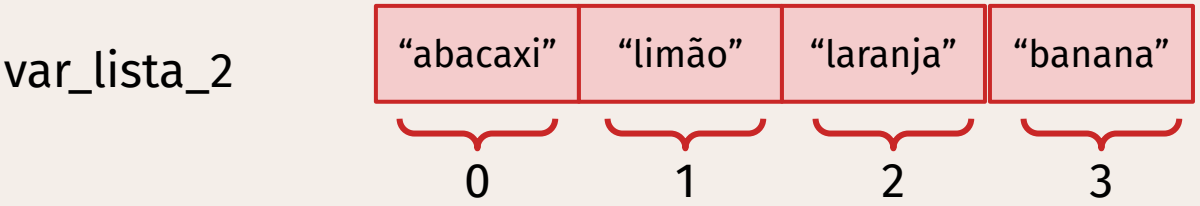
```
var_lista_2[0] = “manga”
print(var_lista_2)
```



Remoção em Listas

Também podemos deletar um elemento indicando o mesmo através de seu índice e utilizando o comando *del*. O resultado na lista, nesse caso, será equivalente à utilização da função *remove*

```
del var_lista_2[0]
print(var_lista_2)
```



Prática em Listas

Faça um programa que receba, através da entrada padrão, as seguintes respostas do usuário:

a) Qual o seu primeiro nome?
b) Qual o seu último sobrenome?
c) Qual a sua idade? (*casting* para int)
d) Qual a sua altura em metros? (*casting* para float)

Armazene os dados recebidos **em ordem em uma lista cadastral** atribuída em uma variável chamada **var_cadastro**. Exiba a lista cadastral na tela identificando cada um dos seus respectivos campos

Listas Aninhadas

Listas também podem ser elementos de listas! Ou seja, é possível armazenar uma lista em outra lista

```
var_lista_1 = []
var_lista_2 = [1, 2, 3, 4]
var_lista_1.append(var_lista_2)
print(var_lista_1, var_lista_2)
```

Listas como Referência

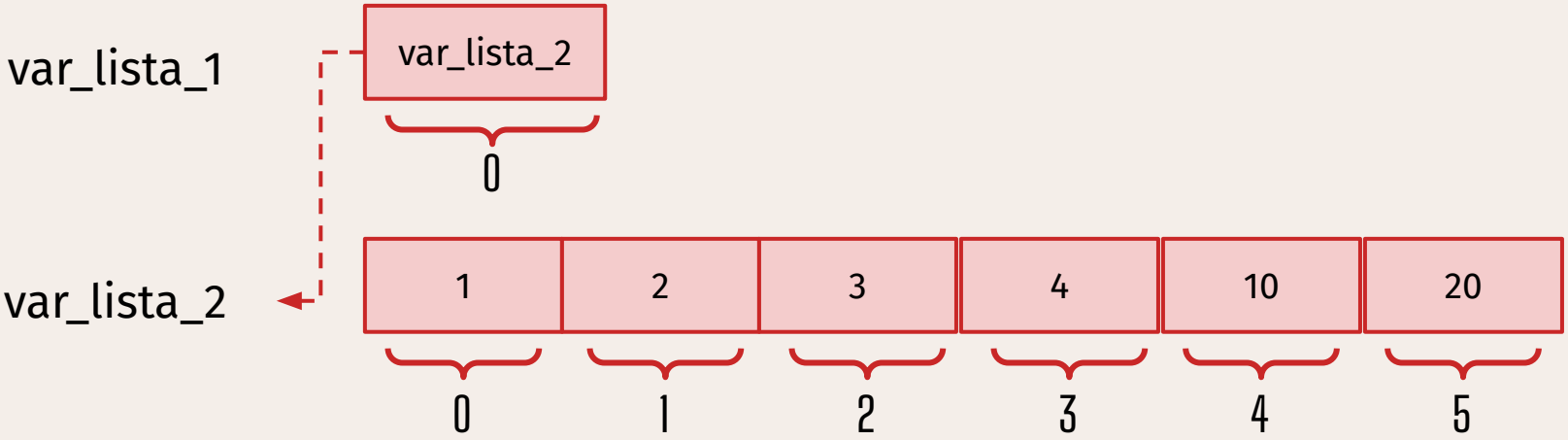
Um ponto muito importante: Python realiza atribuição em variáveis via referência

Ou seja, se modificarmos a lista em um lugar em que ela aparece, modificamos em todos os lugares em que ela é referenciada

```
var_lista_2.append(10)
print(var_lista_1, var_lista_2)
var_lista_1[-1] .append(20)
print(var_lista_1, var_lista_2)
```

Listas como Referência

Esse tipo de relação se dá devido aos **ponteiros ocultos existentes em Python**: não precisamos trabalhar com os ponteiros em Python; mas isso não quer dizer que eles não existam!



Prática em Listas

Considere o formulário da última prática. Esse será o formulário preenchido quando o usuário escolher no **menu principal do programa a opção “c” (cadastrar)**. Cada cadastro realizado deverá ser **armazenado, em ordem de criação, em uma lista** que servirá como nossa base de dados

Além disso, o menu principal também deve oferecer a **opção “e” (exibir)**. Essa opção **apresenta todos os cadastros presentes na base** de dados em ordem de inserção, sendo um registro mostrado em cada linha

Exercício #13

Em uma competição de salto em distância, cada atleta tem direito a cinco saltos. O resultado do atleta será determinado pela **média dos cinco valores informados**. Você deve fazer um programa que receba **o nome e as cinco distâncias, em metros (float), alcançadas pelo atleta** em seus saltos e depois **informe o nome, os saltos e a média dos saltos**. O programa deve ser encerrado quando não for informado o nome do atleta

Fundamentos de Programação

Aula 13

Obrigado e ATÉ A PRÓXIMA!

©Prof. Vinicius Fulber-Garcia - 2025