Aula 6: Listas

Luís Felipe

UFF

15 de Setembro de 2025

Luís Felipe

15/09/25

Vetores

Vetores são construções de linguagens de programação que servem para armazenar vários dados de forma simplificada.

Intuito: Definir variável que armazene mais de um valor.

- Ex.: Suponha que desejemos guardar notas de 100 alunos. Criar 100 variáveis distintas não é uma solução elegante. problema.

Essas variáveis são conhecidas como variáveis compostas, variáveis subscritas, variáveis indexáveis ou arranjos (array = vetores).

Em Python existem três tipos principais de variáveis compostas (com a mesma lógica), cada uma com suas características especiais:

- Listas (foco de hoje)
- Tuplas
- Dicionário

Luís Felipe

15/09/25

Listas em Python

- Uma lista em Python é uma estrutura que armazena vários dados, que podem ser de um mesmo tipo ou não.
- O acesso a um dado específico da lista é feito por meio da sua posição (índice).
- Uma lista é criada com a construção:

  `lista = [dado1, dado2, ..., dadon]`.

  Obs.: Declara-se uma lista, colocando entre colchetes uma sequência de dados separados por vírgula.
- Os índices começam em `0`, pois essa convenção é herdada de linguagens como C, onde o índice representa a distância (ou deslocamento) do primeiro elemento da lista.

Exemplos:

```
>>> a = [1, "ola", 2]
>>> type(a)
<class 'list'>
>>> a[0]
1
>>> a[1]
'ola'
>>> a[2]
2
```

```
>>> a = [1, 4, 2]        # Lista de inteiros
```

```
>>> a = [1, "a", 2]      # Lista c/ tipos distintos
```

```
>>> a = [1, [4,5], [2]] # Lista com outras listas
```

```
>>> a = []               # Lista vazia
```

Luís Felipe

15/09/25

Usando uma lista

- Pode-se acessar uma posição de uma lista usando um índice inteiro.
- Se n é o tamanho da lista, índices válidos vão de 0 a $n - 1$.
- Primeira posição: índice 0
- Última posição: índice $n - 1$
- Sintaxe: `identificador[posicao]`

```
1  notas = [4.5, 8.6, 9, 7.8, 7]
2  print(notas[1] + 2)        # 10.6
3  notas[3] = 0.4
4  print(notas)               # [4.5, 8.6, 9, 0.4, 7]
```

Luís Felipe

15/09/25

Acesso usando variáveis como índice

- É comum usar `for` para percorrer listas.

```
notas = [4.5, 8.6, 9, 7.8, 7]
for i in range(5):
    print(notas[i])

# Saída:
# 4.5
# 8.6
# 9
# 7.8
# 7
```

Exemplo:

```
l = [0,0,0,0,0,0,0,0,0,0]
for i in range(10):
    l[i] = 5 * i
print(l)

# Saída:
# [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Luís Felipe

15/09/25

Listas – Índices negativos e erros

- Índices negativos contam de trás para frente.
- -1 é o último elemento, -2 o penúltimo, etc.
- Acessar uma posição inexistente gera erro `IndexError`.

```
1  notas = [4.5, 8.6, 9, 7.8, 7]
2
3  print(notas[-1])                # 7
4
5  print(notas[100])               # IndexError: list index out of range
6
7  print(notas[-6])                # IndexError: list index out of range
```

Luís Felipe

15/09/25

Listas – Índices e Slicing

- Listas em Python suportam **slicing**, operação que obtém uma sublista.
- Forma geral: `identificador[ind1:ind2]`
- Retorna elementos do índice `ind1` até `ind2 - 1`.

```
1  notas = [4.5, 8.6, 9, 7.8, 7]
2
3  print(notas[1:4])                # [8.6, 9, 7.8]
```

Observações:

- O índice inicial é incluído, o final é excluído.
- Se ind1 for omitido, começa do início da lista.
- Se ind2 for omitido, vai até o final.

```
1  notas = [4.5, 8.6, 9, 7.8, 7]
2
3  print(notas[2:])                 # [9, 7.8, 7]
4  print(notas[:2])                 # [4.5, 8.6]
```

Luís Felipe

15/09/25

Operações básicas com listas

Listas podem ser modificadas. Pode-se incluir incluir e remover itens.

Seja uma lista com nome lista.

- `append(valor)`: adiciona valor ao final da lista.
- `len(lista)`: retorna o número de elementos da lista.
- `valor in lista`: retorna True se valor estiver na lista, caso contrário, False.
- `remove(valor)`: remove a primeira ocorrência de valor na lista.
- `pop()`: remove o último elemento da lista.
  `pop(indice)`: remove o elemento da posição indice.

Exemplo de uso:

```
1  lista = [10, 20, 30]
2  lista.append(40)      # [10, 20, 30, 40]
3  print(len(lista))     # 4
4  print(20 in lista)    # True
5  lista.remove(10)      # [20, 30, 40]
6  x = lista.pop()       # x = 40, lista = [20, 30]
```

```
1  lista = [10, 20, 30]
2
3  lista.insert(1, 40)
4  print(lista)          # [10, 40, 20, 30]
```

Luís Felipe

15/09/25

Funções

É muito comum usar a função len junto com o laço for para percorrer todas as posições de uma lista:

```
1  x = [1,2,3,4,5,6]
2  for i in range(len(x)):
3      print(x[i]+1, end = " ")
4  print("\n")                          # Saída: 2 3 4 5 6 7
```

É comum acrescentar um item no final de uma lista, feito com append:

```
1  x = [6, 5]
2  x.append(98)
3  print(x)                             # Saída: [6, 5, 98]
```

Observação:

- O formato é `lista.append(item)`
- A lista que será modificada vem antes, seguida de ponto, depois o método append com o item como argumento.
- Esse tipo de operação associada a um objeto é chamado de **método**.

Luís Felipe

15/09/25

Preenchendo uma lista

Lendo dados e adicionando com append:
Leia um número inteiro N que indica quantas notas serão digitadas. Em seguida, leia N valores reais (um por vez) e armazene cada valor ao final da lista x usando o método append. Ao término da leitura, imprima a lista x preservando a ordem de entrada dos dados.

Observação: a mensagem "Entre com a nota i" utiliza o índice i iniciando em 0 (as notas são solicitadas nas posições $0, 1, 2, \ldots, N-1$).

```
1  x = []
2  n = int(input("Entre com o numero de notas: "))
3  for i in range(n):
4      dado = float(input("Entre com a nota " + str(i) + ": "))
5      x.append(dado)
6  print(x)
```

Importante: Note que a lista deve ser criada antes de utilizar o append.

Luís Felipe

15/09/25

Concatenação de listas

- Usamos o operador + para juntar duas listas.
- Isso gera uma nova lista sem modificar as originais.

```
1  lista1 = [1, 2, 4]
2  lista2 = [27, 28, 29, 30, 33]
3  x = lista1 + lista2
4  print(x)
5
6  # [1, 2, 4, 27, 28, 29, 30, 33]
```

Luís Felipe

15/09/25

Repetição de listas

- O operador * repete a concatenação de uma lista.

```
1  x = [1, 2]
2  y = 4 * x
3  print(y)
4
5  # [1, 2, 1, 2, 1, 2, 1, 2]
```

Luís Felipe

15/09/25

Outros métodos em listas

- `lista.insert(i, dado)`: insere dado para que fique na posição *i*.
- `del lista[i]`: remove elemento da posição *i*.

```
1  x = [40, 30, 10, 40]
2
3  x.insert(1, 99)              # [40, 99, 30, 10, 40]
4
5  del x[2]                     # [40, 99, 10, 40]
```

Luís Felipe

15/09/25

Elementos Iguais – Versão 1

- Ler dois vetores com 5 inteiros cada.
- Checar quais elementos do segundo vetor são iguais a algum elemento do primeiro vetor.
- Se não houver elementos em comum, o programa deve informar isso.

```
1  x = []
2  y = []
3  for i in range(5):
4      x.append(int(input(f"Valor {i+1} vetor 1: ")))
5      y.append(int(input(f"Valor {i+1} vetor 2: ")))
6
7  umEmComum = False
8  for i in range(len(x)):
9      for j in range(len(y)):
10         if x[i] == y[j]:
11             umEmComum = True
12             print(f"Elemento da pos. {i} do vetor 1 igual elemento da pos. {j} do vetor 2")
13
14 if not umEmComum:
15     print("Nenhum elemento em comum")
```

Luís Felipe

15/09/25

Elementos Iguais – Versão 2

- Similar ao programa anterior, mas agora nos laços percorremos as listas diretamente com seus valores, ao invés de se utilizar índices para as listas.

```
1  x = []
2  y = []
3  for i in range(5):
4      x.append(int(input(f"Valor {i+1} vetor 1: ")))
5      y.append(int(input(f"Valor {i+1} vetor 2: ")))
6
7  umEmComum = False
8  for a in x:
9      for b in y:
10         if a == b:
11             umEmComum = True
12
13 if not umEmComum:
14     print("Nenhum elemento em comum")
```

Luís Felipe

15/09/25

Leitura e Inicialização de Listas

Exemplo 1: Lendo todos os valores de uma vez, separados por espaço.

```
valores = input("Digite os valores na mesma linha: ").split()
```

Exemplo 2: Lendo um vetor com 10 valores, um por vez.

```
valores = [None] * 10
for i in range(len(valores)):
    valores[i] = input("Digite um valor: ")
```

Alternativa: Criando listas já preenchidas usando **list comprehension**.

```
# Lista com 5 zeros
x = [0 for i in range(5)]        # [0, 0, 0, 0, 0]

# Lista com os 5 primeiros pares
x = [2*i for i in range(5)]      # [0, 2, 4, 6, 8]
```

Luís Felipe

15/09/25

Matriz 2D é um Vetor de Vetores

A declaração abaixo corresponde a uma matriz (2 dimensões) cujo tipo base é **boolean**.

```
celulas = [
    [True, False, False, True, True],
    [True, True, False, True, False]
]
```

- O primeiro índice representa a linha.
- O segundo índice representa a coluna.

Luís Felipe

15/09/25

Acessando elementos em uma Matriz

```
1  celulas = [
2      [True, False, False, True, True],
3      [True, True, False, True, False]
4  ]
5
6  print(celulas[0][0])         # Acessa linha 0, coluna 0
7
8  print(celulas[1][0])         # Acessa linha 1, coluna 0
9
10 print(celulas[1][2])         # Acessa linha 1, coluna 2
```

- `celulas[1][2]` retorna o valor da segunda linha, terceira coluna.
- Neste caso: `False`.

Luís Felipe

15/09/25

Exemplo: Loteria Esportiva

Uma matriz de 13 linhas e 3 colunas para representar a aposta.

```
aposta = [
    [" ", "X", " "],
    ["X", " ", " "],
    [" ", " ", "X"],
    ...
]
```

- aposta → matriz completa.
- aposta[i] → linha i (um vetor com 3 caracteres).
- aposta[i][j] → caractere na linha i, coluna j.

Luís Felipe

15/09/25

Exemplo: Estado do Jogo da Velha

```
# 0 = célula vazia
tabuleiro = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]

tabuleiro[1][1] = 1  # Jogador 1: centro
tabuleiro[0][2] = 2  # Jogador 2: sup. direita
tabuleiro[0][0] = 1  # Jogador 1: sup. esquerda
tabuleiro[2][2] = 2  # Jogador 2: inf. direita
```

Cada atualização altera o estado do `tabuleiro`.