



Aula 01 – Listas e Tuplas + Revisão de Lógica

Listas e Tuplas + Revisão de Lógica

1

Revisar conceitos **fundamentais de lógica** de programação.

2

Introduzir as estruturas de dados **listas e tuplas em Python**.

Nesta aula, vamos solidificar a base do pensamento computacional e expandir nossas ferramentas para manipulação de dados.
Preparados para codificar?

Revisão de Lógica: Variáveis e Entrada/Saída



Declaração de variáveis:

- `int` (integers ou inteiros)
- `float` (decimais)
- `str` (strings ou textos)
- `bool` (booleanos – verdadeiro/falso)



Entrada de dados:

- Função `input()`:
Obtém dados do usuário.

Saída de dados:

- Função `print()`:
Exibe informações.

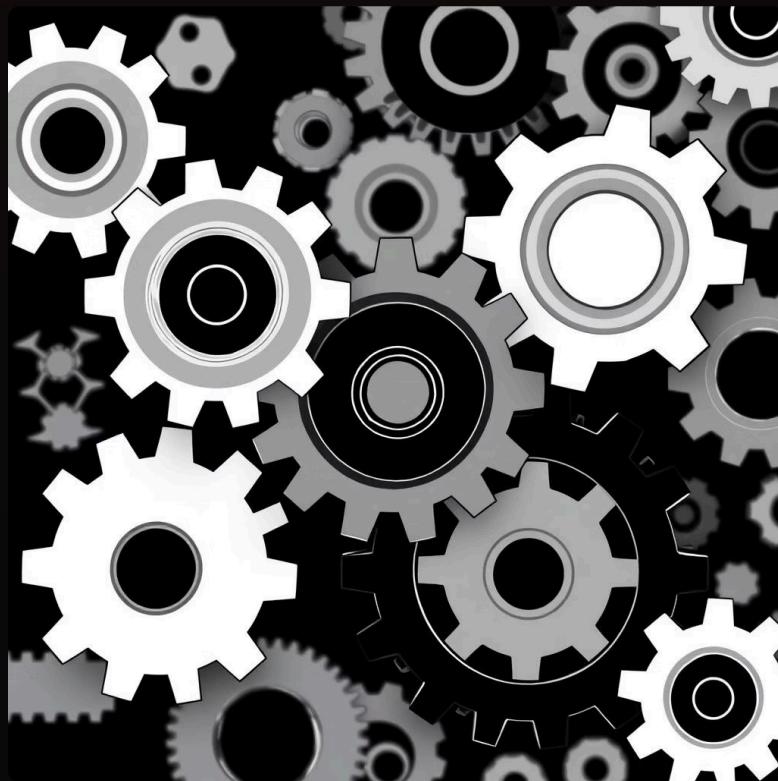
```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))
print(f"Olá, {nome}! Você tem {idade} anos.")
```

Dominar variáveis e as funções de entrada/saída é o primeiro passo para criar programas interativos e funcionais!

Revisão de Lógica: Operadores e Condicionais

• Operadores

- Aritméticos: +, -, *, /, %, //, **
- Relacionais: ==, !=, >, <, >=, <=
- Lógicos: and, or, not



• Condicionais

As condicionais permitem que seu programa tome **decisões** com base em certas condições.

if, elif, else

```
if idade >= 18:  
    print("Maior de idade")  
else:  
    print("Menor de idade")
```

Operadores e condicionais são o coração da lógica de programação, permitindo criar programas que **respondam de forma inteligente** aos dados.

Revisão de Lógica: Estruturas de Repetição

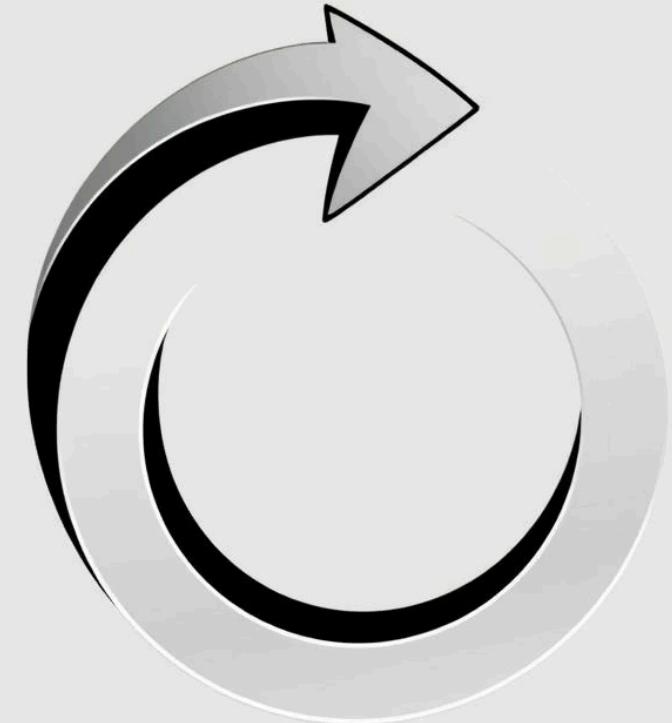
📌 Laço while

Repete um bloco de código **enquanto uma condição for verdadeira**. Ideal para repetições com número incerto de iterações.

```
contador = 1
while contador <= 5:
    print(contador)
    contador += 1
```

ⓘ ⚡ Prática Rápida:

Use o laço **while** para criar contadores personalizados e menus interativos, como um menu de opções para um jogo simples!



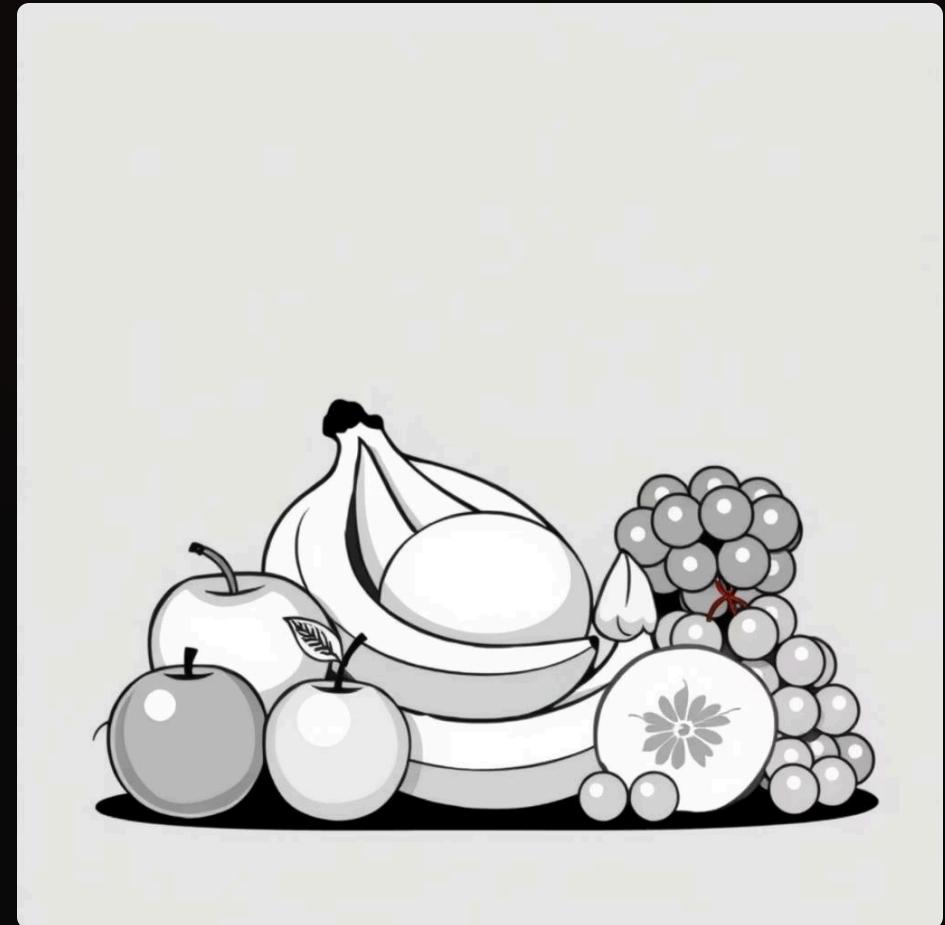
Introdução às Listas

📌 Conceito: Coleção mutável e ordenada

- Armazena múltiplos itens em uma única variável.
- Os itens são **ordenados** (mantêm a sequência de inserção).
- São **mutáveis**, ou seja, seus elementos podem ser alterados, adicionados ou removidos.

✓ Criando listas:

```
lista_vazia = []  
  
lista_vazia1 = list()  
  
frutas = ["maçã", "banana", "uva"]
```



✓ Indexação e fatiamento:

```
print(frutas[0]) # maçã  
print(frutas[1:3]) # ['banana', 'uva']  
# list  
  
print(type(frutas))
```

Listas são como "**caixas organizadoras**" para seus dados, super flexíveis para o dia a dia da programação!

Métodos de Listas

As listas vêm com "poderes" embutidos que facilitam a manipulação dos seus elementos.

- ◆ `append()`

Adiciona um elemento ao final da lista.

- ◆ `insert()`

Adiciona um elemento na posição desejada da lista.

- ◆ `remove()`

Remove a primeira ocorrência de um elemento.

- ◆ `pop()`

Remove o último elemento da lista, posição -1. Ou o elemento na posição especificada.

- ◆ `sort()`

Ordena os elementos da lista de forma crescente.

- ◆ `index()`

Retorna a posição (índice) da primeira ocorrência de um elemento.

- ◆ `count()`

Conta o número de ocorrências de um elemento.

```
numeros = [3, 1, 2]
numeros.append(5) # [3, 1, 2, 5]
numeros.sort() # [1, 2, 3, 5]
print(numeros)
```

Dominar esses métodos torna a manipulação de dados em Python muito mais **eficiente** e **simples**!

Tuplas

📌 Conceito: Coleção imutável

- Também armazena múltiplos itens, mas uma vez criada, **não pode ser alterada**.

✓ Criando tuplas:

```
tupla_vazia = ()
```

```
tupla_vazia1 = tuple()
```

```
cores = ("azul", "verde", "vermelho")
```

```
print(cores[0]) # azul
```

```
print(type(cores)) # tuple
```

✓ Diferenças para Listas:

- **Imutabilidade:** o principal contraste é que você **não pode adicionar, remover ou modificar elementos** após a criação.
- **Segurança:** Mais seguras para dados fixos que **não devem ser alterados acidentalmente** (ex: coordenadas, dados de configuração).
- **Performance:** Podem ser **ligeiramente** mais rápidas para iteração, pois sua estrutura é otimizada.

💡 Iteração:

A iteração (**percorrer os elementos**) funciona da mesma forma com o laço `for` em listas e tuplas.

Tuplas são ideais para quando você precisa de uma coleção de dados que permanecerá **constante**!

Mini-projeto: Cadastro de Alunos

🎯 Objetivo:

Armazenar e gerenciar nomes e notas de alunos usando tudo que aprendemos.

📌 Funcionalidades:

1 Entrada de dados

Coletar nomes e notas dos alunos usando `input()` e armazená-los em **listas**.

2 Cálculo de Médias

Implementar a lógica para calcular a média de cada aluno.

3 Avaliação de Desempenho

Usar **condicionais** (`if/else`) para determinar se o aluno foi aprovado ou reprovado.

4 Apresentação Organizada

Exibir os resultados de forma clara e formatada usando `print()` e `f-strings`.

