

Aula 3: Estrutura de Repetição While

Aprendendo a utilizar um laço de repetição

Repetição com Loops While

Nesta aula, vamos mergulhar no universo dos laços de repetição, essenciais para escrever códigos eficientes e dinâmicos.

1

Necessidade dos Laços

Entender por que precisamos de estruturas que repetem ações.

2

Estrutura While

Compreender como o `while` funciona e onde aplicá-lo.

3

Contadores e Acumuladores

Aprender a usar variáveis para contar e somar dentro dos loops.

4

Prevenção de Loops Infinitos

Garantir que seus laços sempre tenham um fim seguro.

O que é um Loop?

Um **loop** é um bloco de código que se repete múltiplas vezes, economizando linhas e tornando seu programa mais conciso.



- Um **loop** (ou laço de repetição) é uma estrutura fundamental que permite executar um conjunto de instruções repetidamente.
- Ele **evita a repetição manual de código**, tornando-o mais limpo e eficiente.
- Imagine que você precisa mostrar os números de 1 a 1000. Sem loops, você teria que escrever 1000 linhas de `print()`!

Estrutura do While

A estrutura while é um laço de repetição que executa um bloco de código **enquanto uma condição específica for verdadeira**.

Sintaxe:

```
while condição:  
    # bloco de código a ser executado  
    repetidamente
```

O bloco de código deve ser **indentado (TAB)**.

Exemplo Simples:

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

Este código irá imprimir os números de 1 a 5, um por linha.

A **condição** é avaliada a cada início de repetição. Se for verdadeira, o bloco é executado; se for falsa, o laço é encerrado.

Condições de Parada: Evitando Loops Infinitos

Todo loop `while` precisa de uma **condição de saída clara** para evitar que ele execute indefinidamente.



Risco de Loop Infinito

Se a condição do `while` nunca se tornar falsa, o programa ficará preso em um **loop infinito**.



Condição de Saída

A variável usada na condição deve ser **modificada dentro do loop** para que, em algum momento, a condição se torne falsa.



Palavra-chave `break`

Podemos usar `break` para **forçar o encerramento do loop** a qualquer momento, independente da condição principal.

Exemplo com Condição de Parada:

```
numero = 0
while numero < 3:
    print("Executando...")
    numero += 1 # Aumenta o número, eventualmente tornando a condição falsa
```

Contadores: Rastreamento Repetições

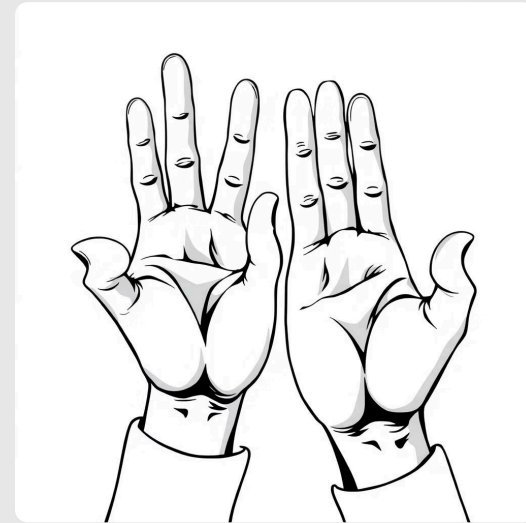
Um **contador** é uma variável numérica que tem seu valor alterado (geralmente incrementado ou decrementado) a cada iteração de um loop. É usado para **saber quantas vezes uma ação ocorreu**.

Definição:

Variável iniciada com um valor e alterada por um valor fixo (ex: +1, -1) dentro do loop.

Utilidade:

- Controlar o número de repetições.
- Acompanhar a posição atual em uma sequência.
- Limitar a execução de uma tarefa.



Exemplo:

```
contador = 1
while contador <= 10:
    print("Contagem:", contador)
    contador += 1 # Incrementa o contador em 1
```

Este loop contará de **1 a 10**.

Acumuladores: Somando Valores

Um **acumulador** é uma variável que **armazena a soma ou o resultado de operações sucessivas** realizadas em cada iteração de um loop. Diferente do contador, que soma um valor fixo, o acumulador soma valores variáveis.



Definição:

Variável iniciada com um valor neutro (geralmente 0 para soma, 1 para produto) e modificada somando (ou subtraindo/multiplicando) outros valores.

Utilidade:

- Calcular somas de séries de números.
- Totalizar valores (ex: compras no carrinho).
- Consolidar resultados de múltiplas operações.

Exemplo:

```
soma = 0 # Acumulador
numero = 1
while numero <= 5:
    soma += numero # Acumula o valor de 'numero'
    numero += 1
print("Soma total:", soma) # Resultado: 15 (1+2+3+4+5)
```

Condicionais dentro do While: Decisões Dinâmicas

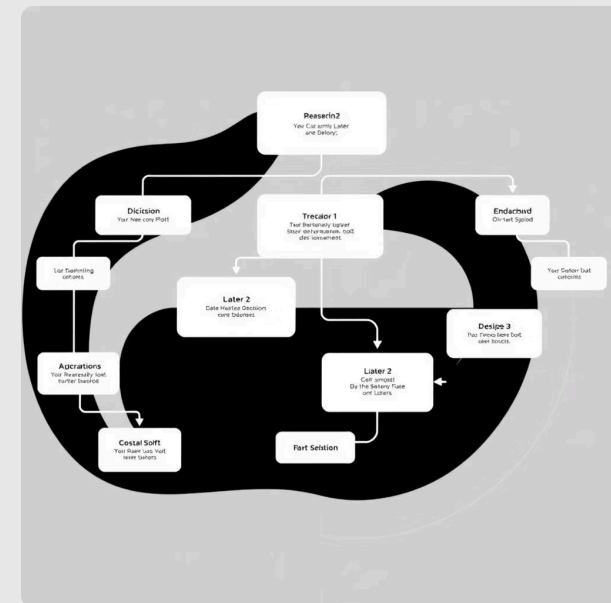
É comum e muito útil combinar laços de repetição com estruturas condicionais (if, elif, else). Isso permite que seu programa **tome decisões e execute diferentes blocos de código** a cada iteração do loop, dependendo das condições.

Flexibilidade:

Cada repetição pode ter um comportamento distinto com base em variáveis ou entradas do usuário.


Exemplo de Uso:

- Verificar se um número é par ou ímpar.
- Processar itens de uma lista de diferentes maneiras.
- Responder a inputs do usuário durante a execução.



Exemplo:

```
numero = 1
while numero <= 5:
    if numero % 2 == 0:
        print(numero, "é par")
    else:
        print(numero, "é ímpar")
    numero += 1
```


A 3D illustration of a maze with a central path leading to a door labeled 'break'. The maze is constructed from white, blocky walls on a dark grey floor. The path leads to a white door with a sign that says 'break'. The background is dark grey.

While True + Break: Laços Indefinidos

A construção `while True` cria um loop que, por definição, é **infinito** (pois `True` nunca é falso). Para sair desse loop, você deve usar a palavra-chave `break` quando uma condição específica for atingida dentro do bloco de código. É ideal para situações onde o número de repetições não é conhecido antecipadamente, como menus interativos ou leituras de entrada do usuário.

Exemplo:

```
while True: # O loop rodará indefinidamente
    resposta = input("Digite 'sair' para encerrar: ")
    if resposta == "sair":
        print("Programa encerrado!")
        break # Sai do loop quando a condição é atendida
    else:
        print("Você digitou:", resposta)
```

Hora da Prática!

Agora, é a sua vez de colocar a mão na massa e aplicar o que aprendeu!

1 Contagem Simples

Crie um programa que mostre uma contagem de **1 a 10** usando `while`.

2 Contagem Personalizada

Desenvolva um programa que conte de **1 até um valor limite** definido pelo usuário.

3 Contagem Regressiva de Ano Novo

Faça um programa de contagem regressiva de **10 até 1** e exiba a mensagem "**Feliz ano novo!**" ao final.

4 Par ou Ímpar Interativo

Crie um programa que permita ao usuário **digitar números**, identifique se são **par ou ímpar**, e só encerre quando o usuário digitar **0**.