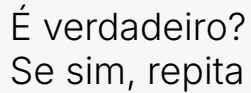


O que são laços de repetição?

Laços são estruturas que permitem **executar um bloco de código repetidamente**, enquanto uma condição for verdadeira. Estruturas de laços que temos em Python:

- For
- While

A leitura de um laço pode ser interpretada como: **repita até que** a condição se torne falsa.



É verdadeiro?
Se sim, repita

Sintaxe do laço FOR

O laço **for** é utilizado para executar um bloco de código para cada elemento do iterável. Ele é útil quando você **sabe exatamente quantas iterações** deseja realizar.

for **elemento** **in** **iterável** :

bloco de código

Os dois pontos e a palavra **in** são obrigatórios.

É uma coleção de valores, como listas, tuplas, dicionários, dentre outros.

Cada item individual do iterável que é processado dentro do laço.

Exemplo de código FOR

Esse exemplo percorre uma lista de nomes e imprime na tela cada um deles.

```
main.py  
  
nomes = ["Carlos", "Ana", "Pedro", "Maria"]  
for nome in nomes:  
    print(nome)
```

Primeiro, criamos uma lista chamada `nomes`, que contém os nomes: `["Carlos", "Ana", "Pedro", "Maria"]`

O termo `"nome"` é apenas um identificador que usamos para representar cada elemento da lista enquanto percorremos com o `laço for`. Você pode substituir `"nome"` por qualquer outro nome de variável que faça sentido para o seu contexto.

Em seguida, utilizamos um **laço for** para percorrer todos os nomes na lista e imprimir cada nome na tela.

Sintaxe do laço WHILE

O laço **while** executa um bloco de código enquanto uma condição especificada for **verdadeira**. Ele é útil quando você **não sabe exatamente quantas iterações** serão necessárias, já que a execução depende do resultado da condição a cada iteração.

while **condicao** :
 bloco de código

A condição é qualquer expressão que resulte em um valor booleano (*True* ou *False*). Exemplo: *while numero < 5:*

Operadores de comparação

Os operadores de comparação são utilizados para comparar valores, retornando **True** ou **False**, dependendo da condição estabelecida.

Operador	Conceito	Exemplo
> (Maior que)	Verifica se um valor é maior que outro	<code>x > 10</code>
< (Menor que)	Verifica se um valor é menor que outro	<code>x < 10</code>
== (Igual a)	Verifica se um valor é igual a outro	<code>x == 10</code>
!= (Diferente de)	Verifica se um valor é diferente de outro	<code>x != 10</code>
>= (Maior ou igual a)	Verifica se um valor é maior ou igual a outro	<code>x >= 10</code>
<= (Menor ou igual a)	Verifica se um valor é menor ou igual a outro	<code>x <= 10</code>

Exemplo de código WHILE

Esse exemplo percorre uma lista de nomes e imprime na tela cada um deles.

```
main.py

contador = 0

while contador < 5:
    print(f"Contador atual: {contador}")
    contador += 1
```

Dentro do laço, imprimimos o valor atual do contador e, em seguida, incrementamos contador em 1 com `contador += 1`. Esse processo se repete até que a condição `contador < 5` se torne falsa (ou seja, quando contador chegar a 5).

Definimos uma variável contador com o valor 0. O contador serve para **garantir que a condição eventualmente se torne falsa.**

O laço while verifica a condição `contador < 5`. **Enquanto essa condição for verdadeira**, o bloco de código dentro do while será executado.

Loop Infinito

Um loop infinito é quando um laço de repetição continua a **executar sem parar**, porque a condição para sair dele nunca se torna falsa.

```
main.py  
  
contador = 0  
  
while contador < 5: ◀ .....  
    print("Contador:", contador)
```

Neste exemplo, o laço continuará para sempre imprimindo "Contador: 0", porque a condição `contador < 5` nunca mudará para falsa. Isso pode ocorrer tanto em laços `for` quanto `while`.

Break

Mas e se quisermos interromper a execução de um laço antes que ele termine naturalmente? É aí que a instrução `break` entra em cena. O `break` permite que você saia imediatamente de um laço, mesmo que a condição para continuar seja verdadeira.

```
main.py

nomes = ["PM3", "Alura", "Latam", "Outros"]

for nome in nomes:
    if nome == "Alura":
        print("Nome encontrado! Saindo do laço.")
        break
    print(nome)
```

Neste exemplo, o laço irá percorrer os nomes e, ao encontrar "Alura", imprimirá "Nome encontrado! Saindo do laço." e sairá do laço, não imprimindo os nomes que vêm a seguir.

Continue

Agora, e se quisermos pular a execução de uma iteração específica do laço, mas continuar com as demais? O `continue` permite que você pule para a próxima iteração do laço, ignorando o restante do código na iteração atual.

```
main.py

nomes = ["PM3", "Alura", "Latam", "Outros"]

for nome in nomes:
    if nome == "Alura":
        print("Ignorando Alura.")
        continue
    print(f"Nome: {nome}")
```

Neste exemplo, quando o laço encontra "Alura", ele imprime "Ignorando Alura." e salta a impressão do nome "Alura", continuando com os outros nomes na lista.

Funções úteis em laços

- **len()** é utilizada para obter o comprimento de uma lista, string ou outro tipo de coleção. Ela nos permite saber quantas iterações precisamos realizar em um laço.
- **range()**, gera uma sequência de números, que é frequentemente utilizada para controlar a iteração em laços for. Com ela, podemos especificar um intervalo de números para iterar, podendo também definir um passo. Por exemplo, `range(6)` gera os números de 0 a 5, permitindo que o laço for execute cinco vezes.