

Estruturas de Repetição em Python

1) Por que precisamos de repetições?

Na vida real, muitas tarefas são **repetitivas**:

- Tomar remédio **3 vezes ao dia**.
- Contar **quantos passos** foram dados em uma caminhada.
- Enviar **mensagens automáticas** para vários contatos.

Se fôssemos escrever um programa que imprime os números de 1 a 100 sem usar repetições, teríamos que escrever **100 comandos print**, um para cada número! Isso seria cansativo, repetitivo e nada prático.

Para resolver isso, usamos as **estruturas de repetição (loops)**, que permitem ao computador **executar um bloco de código várias vezes automaticamente**.

No Python, temos principalmente dois tipos de repetição:

- **while** → repete **enquanto uma condição for verdadeira**.
 - **for** → repete **um número definido de vezes** ou percorre uma sequência.
-

2) O while: repetindo até a condição ser falsa

O comando `while` significa **“enquanto”**. Ele serve para repetir um bloco de código **sem saber de antemão quantas vezes a repetição vai acontecer**.

A ideia é:

- Enquanto a condição for **verdadeira (True)** → o programa continua repetindo.
 - Quando a condição se tornar **falsa (False)** → o loop **para**.
-

♦ Quando usar o while?

O `while` é mais adequado quando:

- Não sabemos **quantas vezes** vamos repetir a ação.
- A repetição depende de uma **condição que pode mudar ao longo da execução**.
- Queremos que o usuário ou algum cálculo **controle o fim do loop**.

Em resumo: **use while quando a repetição depende de uma condição lógica e não de um número fixo de vezes.**

♦ Estrutura básica

```
while <condição>:  
    bloco
```

- <condição> → uma pergunta que resulta em **True** ou **False**.
- bloco → instruções que serão repetidas enquanto a condição for **True**.

Importante: a condição é testada **antes de cada repetição**. Se começar **falsa**, o bloco nem chega a ser executado.

♦ Exemplo 1 – contador simples

```
count = 0  
print("Iniciando...")  
  
while count < 10:  
    print(count, end=" ")  
    count += 1 # soma 1 a cada volta  
  
print("\nFim")
```

Saída:

```
Iniciando...  
0 1 2 3 4 5 6 7 8 9  
Fim
```

Explicação:

1. O programa começa com `count = 0`.
 2. Testa: `count < 10?` → `True` → entra no loop.
 3. Imprime o valor e soma 1.
 4. Repete até `count` chegar a 10.
 5. Quando a condição fica `False`, o loop termina.
-

♦ Exemplo 2 – perguntando até o usuário parar

Imagine que queremos perguntar várias vezes se a pessoa deseja continuar:

```
resposta = "s"
```

```
while resposta == "s":  
    print("Bem-vindo ao sistema!")  
    resposta = input("Deseja repetir? (s/n): ")
```

- Enquanto o usuário digitar "s", a mensagem será mostrada.
- Se digitar "n", a condição fica falsa e o loop acaba.

Esse tipo de uso é muito comum quando **o número de repetições depende do usuário**.

Resumo

- O while é usado quando **não sabemos quantas vezes o código precisa se repetir**.
 - Ele repete **enquanto a condição for verdadeira**.
 - Se a condição for falsa desde o início, o bloco **não roda nenhuma vez**.
 - Muito útil em **interações com o usuário** ou em cálculos que só terminam quando atingem uma meta.
-

3) O for: repetindo um número definido de vezes

Em muitas situações, nós já sabemos **quantas vezes queremos repetir uma tarefa**. Exemplo da vida real:

- Um médico pede que o paciente tome **um remédio 3 vezes ao dia**.
- Uma professora manda os alunos fazerem **10 repetições de um exercício**.
- Um pesquisador precisa calcular a média de **100 entrevistas de idosos**.

Se o número de repetições é **conhecido de antemão**, o comando **for** é a escolha ideal.

♦ Estrutura do for no Python

```
for <variável> in range(início, fim, passo):  
    bloco
```

- **início** → o valor inicial da contagem (por padrão é 0).
- **fim** → até onde vai (o último número **não é incluído**).
- **passo** → de quanto em quanto a contagem aumenta (por padrão é 1).

Esse `range()` gera uma sequência de números para o `for` percorrer.

♦ Exemplo 1 – contar de 0 até 9

```
print("Contando com for:")
for i in range(0, 10):
    print(i, end=" ")
print("\nFim")
```

Saída:

```
Contando com for:
0 1 2 3 4 5 6 7 8 9
Fim
```

- Aqui, o `for` começa no **0** e vai até **9** (o 10 não é incluído).
 - A cada volta, a variável `i` recebe o próximo número da sequência.
-

♦ Exemplo 2 – contando de 2 em 2

Podemos mudar o **passo** para controlar como a contagem avança:

```
for i in range(0, 10, 2):
    print(i, end=" ")
```

Saída:

```
0 2 4 6 8
```

Isso é útil, por exemplo, quando queremos analisar **apenas registros pares** em uma lista ou somar valores de 2 em 2.

♦ Exemplo 3 – repetição sem precisar do número

Às vezes, queremos apenas repetir uma ação várias vezes, sem usar a variável de controle. Nesse caso, usamos o underline `_` como nome “anônimo”:

```
for _ in range(5):
    print("Elefante!")
```

Saída:

Elefante!
Elefante!
Elefante!
Elefante!
Elefante!

O `_` é um **nome válido de variável**, mas por convenção significa: “não me importo com esse valor”.

♦ Quando usar o for?

Use o for quando:

- O número de repetições é **conhecido** (ex.: percorrer 100 entrevistas).
- Você quer **percorrer uma sequência de dados** (listas, nomes, idades, respostas de questionários etc.).
- Precisa deixar o código **curto e legível**, evitando aninhamentos complicados.

Resumo

- O for percorre **intervalos ou coleções de dados**.
- Usa a função `range()` para definir início, fim e passo.
- É ideal quando sabemos **quantas vezes queremos repetir**.
- Pode ser usado com variáveis descritivas ou `_` quando o valor não importa.

4) Interrompendo um loop: break

Quando usamos um loop (for ou while), ele normalmente **segue até o fim da sequência** ou até que a condição fique **falsa**. Mas em algumas situações, não queremos esperar o final: precisamos **parar no meio do caminho**.

Para isso, usamos o comando **break**.

♦ Como funciona?

- O break **interrompe imediatamente** o loop em que ele está.
 - O programa “pula” para a **primeira linha após o loop**.
 - Não importa se ainda havia repetições a serem feitas — o loop é encerrado na hora.
-

♦ Estrutura

```
for <variável> in sequência:
    if <condição>:
        break    # encerra o loop
    # outras instruções
```

Ou com while:

```
while <condição>:
    if <condição-de-parada>:
        break
    # outras instruções
```

♦ Exemplo 1 – parar em um número específico

```
num = int(input("Digite um número para parar o loop: "))

for i in range(0, 6):
    if i == num:
        break
    print(i, end=" ")

print("\nFim")
```

- Se o usuário digitar 3, o loop para quando `i == 3`.
- A saída será:

```
0 1 2
Fim
```

♦ Exemplo 2 – procurar um valor em uma lista

Imagine que temos as idades de idosos em um grupo e queremos saber **se existe alguém com 100 anos**. Assim que encontrarmos, não precisamos continuar procurando.

```
idades = [63, 72, 81, 90, 100, 87, 75]

for idade in idades:
    if idade == 100:
        print("Encontramos um idoso com 100 anos!")
        break
    print("Verificando idade:", idade)

print("Fim da busca")
```

Saída:

```
Verificando idade: 63
Verificando idade: 72
Verificando idade: 81
Verificando idade: 90
Encontramos um idoso com 100 anos!
Fim da busca
```

O loop **parou assim que achou o valor 100**, sem precisar verificar os outros.

◆ Exemplo 3 – repetição controlada pelo usuário

Podemos usar `break` para sair de um loop `while` quando o usuário pedir:

```
while True: # loop infinito
    resposta = input("Deseja continuar? (s/n): ")
    if resposta == "n":
        print("Encerrando...")
        break
    print("Você escolheu continuar!")
```

- O `while True` cria um loop infinito.
 - O `break` é a “porta de saída” — só para quando o usuário digitar “n”.
-

◆ Quando usar `break`?

Use `break` quando:

- Quer **encerrar o loop imediatamente** ao encontrar o que procura.
- Precisa de uma **condição de saída extra** além da do próprio loop.
- Está usando um `while True` e deseja que o usuário ou outro evento decida o momento de parar.

Cuidado: não abuse do break. Se usado sem critério, pode deixar o programa confuso e difícil de entender.

Resumo

- break serve para **interromper um loop na hora**.
 - É muito útil em buscas, interações com usuário e condições de parada inesperadas.
 - Sempre que usamos break, o programa continua a execução **logo após o loop**.
-

5) Pulando uma repetição: continue

O comando **continue** também altera o fluxo de um loop, mas de uma forma diferente do break.

- O **break para o loop inteiro** na hora.
 - O **continue não encerra o loop**, apenas **pula a repetição atual** e segue para a próxima.
-

♦ Quando usar continue?

Use continue quando:

- Você quer **ignorar alguns casos específicos** dentro de um loop.
 - Quer que o programa **não execute o bloco completo** para determinados valores.
 - Precisa “filtrar” dados, analisando apenas os que interessam.
-

♦ Exemplo 1 – imprimir apenas números pares

```
for i in range(0, 10):  
    if i % 2 == 1:  
        continue # se for ímpar, pula para a próxima volta  
    print(i, "é par")
```

Saída:

0 é par
2 é par
4 é par
6 é par
8 é par

Aqui, sempre que *i* é ímpar, o `continue` faz o Python **pular o `print`** e ir direto para a próxima repetição.

♦ Exemplo 2 – filtrando respostas de um questionário

Um grupo de pesquisa perguntou a vários idosos **quantos minutos de caminhada por dia** eles fazem. Alguns, porém, deixaram a resposta em branco (valor 0). Queremos imprimir **apenas os que responderam com valor positivo**.

```
minutos = [0, 20, 35, 0, 50, 10, 0]

for m in minutos:
    if m == 0:
        continue # ignora respostas vazias
    print("Idoso registrou:", m, "minutos de caminhada")
```

Saída:

```
Idoso registrou: 20 minutos de caminhada
Idoso registrou: 35 minutos de caminhada
Idoso registrou: 50 minutos de caminhada
Idoso registrou: 10 minutos de caminhada
```

♦ Exemplo 3 – limpar dados inválidos

Suponha que estamos analisando a **idade dos participantes** de um estudo, mas alguns digitam números impossíveis (como -5 ou 200). Queremos processar apenas idades válidas (entre 0 e 120).

```
idades = [72, 65, -5, 80, 200, 91]

for idade in idades:
    if idade < 0 or idade > 120:
        continue # pula valores inválidos
    print("Idade registrada:", idade)
```

Saída:

```
Idade registrada: 72
Idade registrada: 65
Idade registrada: 80
Idade registrada: 91
```

Resumo

- O continue serve para **pular apenas a repetição atual** do loop.
 - Muito útil em **filtragem de dados** (ex.: ignorar respostas inválidas ou vazias).
 - Diferente do break, ele **não encerra o loop**, apenas “salta” para a próxima volta.
-

6) for com else

No Python, um loop for (ou while) pode ter um bloco **else**. Esse else **não significa a mesma coisa que em um if**.

Aqui, o **else só é executado se o loop terminar normalmente**, ou seja:

- Quando **toda a sequência foi percorrida até o fim**.
 - Mas **não é executado** se o loop for interrompido por um break.
-

♦ Estrutura

```
for <variável> in sequência:
    # bloco do for
    if <condição>:
        break
else:
    # bloco do else (só executa se não houve break)
```

♦ Exemplo 1 – procurando um número

```
num = int(input("Digite um número para procurar: "))

for i in range(0, 6):
    if i == num:
        print("Número encontrado!")
        break
    print("Verificando:", i)
else:
    print("Todos os números foram percorridos e não encontramos o")

print("Fim")
```

Saída 1 (usuário digita 3):

```
Verificando: 0
Verificando: 1
Verificando: 2
Número encontrado!
Fim
```

O break foi usado → o else **não executa**.

Saída 2 (usuário digita 7):

```
Verificando: 0
Verificando: 1
Verificando: 2
Verificando: 3
Verificando: 4
Verificando: 5
Todos os números foram percorridos e não encontramos o valor.
Fim
```

O loop chegou ao fim naturalmente → o else **executa**.

◆ Exemplo 2 – procurando um idoso com 100 anos

Imagine que temos uma lista com idades de idosos de um grupo e queremos verificar se **há alguém com 100 anos**.

```
idades = [65, 72, 80, 91, 87]

for idade in idades:
    if idade == 100:
        print("Encontramos um idoso com 100 anos!")
        break
    else:
        print("Não há ninguém com 100 anos no grupo.")
```

Saída:

Não há ninguém com 100 anos no grupo.

♦ Exemplo 3 – validando dados

Se estivermos processando respostas de uma pesquisa e quisermos verificar se **todas as idades estão válidas** (entre 0 e 120 anos):

```
idades = [72, 65, 81, 130, 90]

for idade in idades:
    if idade < 0 or idade > 120:
        print("Valor inválido encontrado:", idade)
        break
    else:
        print("Todos os dados são válidos!")
```

Saída:

Valor inválido encontrado: 130

O break parou o loop, então o else não rodou.

Se a lista fosse [72, 65, 81, 90], o else apareceria:

Todos os dados são válidos!

Quando usar for ... else?

- **Busca em listas ou coleções:** para indicar se o item foi encontrado ou não.
- **Validação de dados:** para confirmar que todos os itens passaram no teste.
- ****Casos em que você quer uma “mensagem final” somente se o loop não foi interrompido.**

Resumo

- O `else` em loops só executa se o loop terminar sem `break`.
- É útil para **buscar, validar ou confirmar resultados**.
- Pode evitar a necessidade de variáveis auxiliares para indicar se algo foi encontrado.

7) Convenções para variáveis de loop

Quando usamos o `for`, precisamos de uma **variável de controle** — ou seja, um nome que vai receber, a cada repetição, o próximo valor da sequência.

♦ Convenções comuns

- Em matemática, usamos muito as letras **i, j, k** para representar índices.
- Esse costume veio de linguagens antigas, como o **Fortran** (década de 1950).
- Por isso, em Python (e em muitas outras linguagens), é **normal encontrar i, j, k em loops**.

Exemplo simples:

```
for i in range(5):  
    print("Repetição número", i)
```

Saída:

```
Repetição número 0  
Repetição número 1  
Repetição número 2  
Repetição número 3  
Repetição número 4
```

♦ Nomes mais descritivos

Em **Ciência de Dados**, é comum lidarmos com listas de idades, nomes, rendas e outros atributos. Nesses casos, é mais **claro** dar nomes que representem o que a

variável contém.

Exemplo:

```
idades = [60, 62, 64, 67]

for idade in idades:
    print("Participante com", idade, "anos")
```

Saída:

```
Participante com 60 anos
Participante com 62 anos
Participante com 64 anos
Participante com 67 anos
```

Perceba como idade é mais **legível** que simplesmente i.

Resumo

- Para contagens simples → use i, j, k.
- Para dados mais complexos (como listas de idosos, nomes, salários, etc.) → use nomes **descritivos** (idade, nome, salário).
- Isso ajuda a **entender melhor o código**, principalmente em projetos de ciência de dados, onde lidamos com muitas variáveis diferentes.

8) Exemplo aplicado: jogo do dado (while com interação)

As estruturas de repetição também são muito úteis para criar **interatividade com o usuário**. Um exemplo clássico é um **jogo de dados**, em que o jogador decide se quer continuar jogando ou parar.

♦ Exemplo: jogo do dado

```
import random

MIN = 1
MAX = 6

resposta = "s" # começamos assumindo que o usuário quer jogar

while resposta == "s":
    print("Rolando os dados...")
    dado1 = random.randint(MIN, MAX)
    dado2 = random.randint(MIN, MAX)
    print("Resultado:", dado1, "e", dado2)
    resposta = input("Deseja jogar novamente? (s/n): ")
```

♦ Como funciona esse programa

1. Definimos os limites do dado: mínimo 1, máximo 6.
 2. O loop começa com `while True` (ou `while resposta == "s"`).
 3. A cada repetição:
 - Dois números aleatórios entre 1 e 6 são sorteados (como se fossem dados reais).
 - O programa mostra os resultados.
 - Pergunta ao usuário se deseja jogar novamente.
 4. Se o usuário digitar "s", o loop continua.
 5. Se digitar "n", a condição do `while` se torna **falsa** e o loop termina.
-

♦ Exemplo aplicado a idosos – registro de pressão arterial

Suponha que você esteja ajudando um grupo de idosos a **registrar suas medidas de pressão arterial**. Eles podem digitar várias medidas enquanto desejarem.

```
pressao = "s"

while pressao == "s":
    sistolica = int(input("Digite a pressão sistólica (ex: 120): "))
    diastolica = int(input("Digite a pressão diastólica (ex: 80): "))

    if sistolica >= 140 or diastolica >= 90:
        print("Pressão alta! Procure orientação médica.")
    else:
        print("Pressão dentro do limite.")

    pressao = input("Deseja registrar outra medida? (s/n): ")

print("Obrigado por registrar suas informações!")
```

- O idoso pode repetir quantas vezes quiser.
- Quando digitar "n", o programa para.

Resumo

- O while pode criar **loops interativos** que só param quando uma condição muda.
 - O continue permite **pular uma repetição específica**.
 - O for é melhor quando sabemos o **número de repetições**; o while é melhor quando **não sabemos** quando o processo deve terminar.
-