

# **CURSO DE PYTHON**



## **PARA INICIANTE**



## AULA 04: LOOPS - ESTRUTURAS DE REPETIÇÃO

Nessa aula você aprenderá o conceito e a aplicação dos **Loops**, estruturas de repetição usadas pelo python para automatizar tarefas repetitivas e economizar tempo, evitando a necessidade de escrever o mesmo código várias vezes.

Também serão apresentados comandos para controlar o uso dos loops

```
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

self.fingerprints = ...
self.logdups = True
self.debug = debug
self.logger = logging
if path:
    self.file = open
    self.file.seek(
    self.fingerprin

@classmethod
def from_settings(cls,
debug = settings.
return cls(job_di

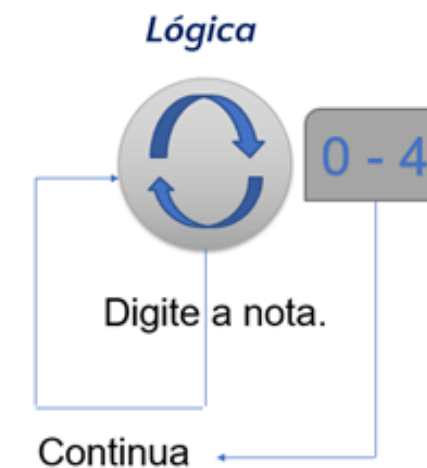
def request_seen(self
fp = self.request
if fp in self.fi
    return True
self.fingerprint
if self.file:
    self.file.w

def request_fingerp
return request
```

# O QUE SÃO LOOPS EM PYTHON?

**Loops** em Python são estruturas de controle de fluxo que permitem a repetição de um bloco de código várias vezes, de acordo com uma condição especificada. Eles são úteis para automatizar tarefas repetitivas, economizando tempo e esforço do programador, além de tornar o código mais eficiente e legível.

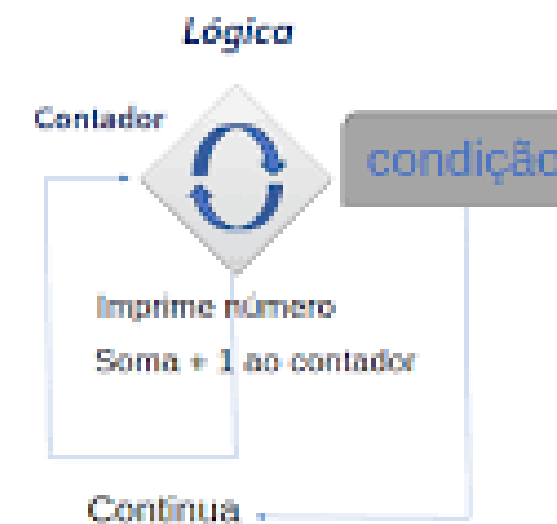
## Laço de repetição - FOR



*Prática*

```
for c in Range(0,4):  
    Digite a nota  
    Continua
```

## Laço de repetição - WHILE



*Prática*

```
C = 0  
while C < 10:  
    print(c)  
    C += 1  
print('Fim')
```

# LAÇO DE REPETIÇÃO - FOR

O loop for é mais flexível em Python do que em linguagens como C ou Java, onde é frequentemente usado com contadores explícitos. Em Python, o for é utilizado para iterar sobre qualquer objeto iterável. Objetos iteráveis incluem listas, tuplas, strings, dicionários, conjuntos, além de geradores e objetos personalizados que implementem o protocolo iterador.

## Sintaxe:

```
for item in sequencia:  
    # código a ser executado para cada item
```

## Exemplo:

```
frutas = ["Maçã", "Banana", "Laranja"]  
for fruta in frutas:  
    print(fruta)  
  
#saída:  
Maçã  
Banana  
Laranja
```

Neste exemplo, o loop percorre a lista de frutas e imprime cada uma delas.



# LAÇO DE REPETIÇÃO - WHILE

O loop while é mais genérico que o for, pois ele continua repetindo enquanto uma condição booleana for verdadeira. Ele pode ser utilizado em cenários onde o número de iterações não é previamente conhecido ou pode variar dinamicamente durante a execução.

## Sintaxe:

```
while condição:  
    #Bloco de código a ser executado
```

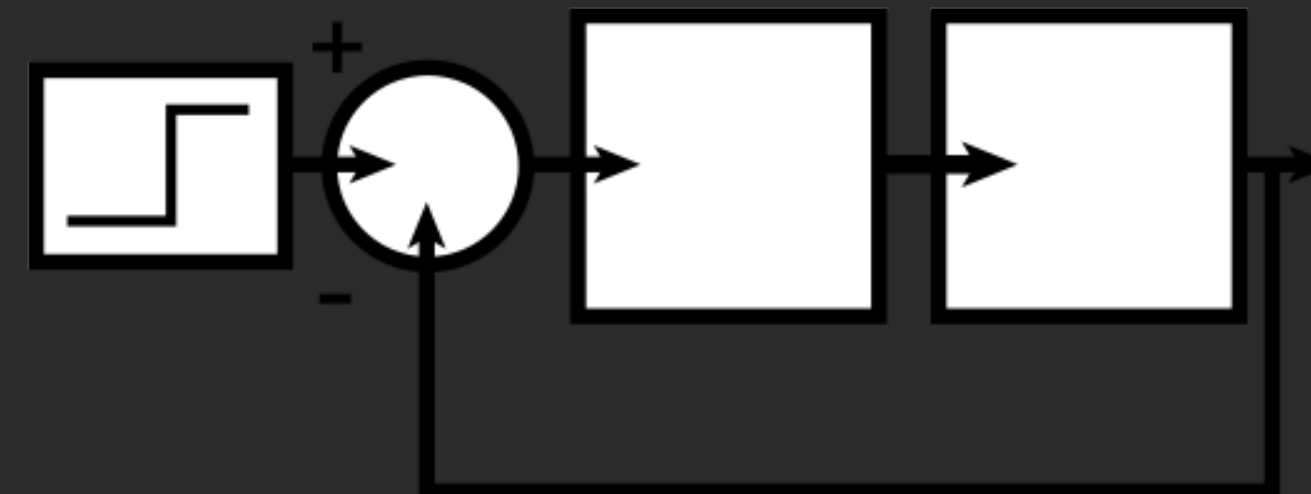
## Exemplo:

```
contador = 0  
while contador < 5:  
    print(contador)  
    contador += 1  
#Saída:  
0  
1  
2  
3  
4
```

O loop while continua até que o valor de contador atinja 5, incrementando-o a cada iteração.

## CONTROLE DE LOOPS

É importante aprofundar os conceitos de como interromper ou modificar o fluxo de execução de loops, utilizando comandos especiais como **break**, **continue**, e a cláusula **else**, que também pode ser usada em loops. Esses controles tornam o uso de loops mais flexível e ajudam a lidar com casos específicos de iteração.



Esses controles de fluxo oferecem ao programador maior controle sobre como e quando um loop deve continuar ou ser encerrado, são importantes para evitar loops infinitos ou desnecessários para melhorar a eficiência do código..


## BREAK

O comando `break` interrompe a execução do loop imediatamente, independentemente de a condição do loop ainda ser verdadeira. É comumente utilizado para interromper um loop quando uma condição específica é atendida antes de o loop completar sua execução natural.

### Aplicações práticas:

- Encontrar um elemento específico em uma lista e parar a busca assim que ele for encontrado.
- Terminar a execução de um programa baseado em uma condição (ex: quando o usuário deseja sair).

## Exemplo



```
for numero in range(10):  
    if numero == 5:  
        break # Interrompe o loop quando o número é 5  
    print(numero)  
  
#saída:  
0  
1  
3  
4
```

Aqui, o loop é interrompido ao encontrar o número 5, e nenhum número adicional é impresso.

## CONTINUE

O comando continue permite que você pule a iteração atual e vá diretamente para a próxima, sem interromper o loop completamente. Ele é útil quando você quer ignorar certas condições ou entradas, mas continuar a execução do loop com os demais elementos.

### Aplicações práticas:

- Ignorar entradas inválidas ou indesejadas durante a execução do loop.
- Pular iterações com base em condições temporárias.

## Exemplo

```
for numero in range(10):  
    if numero % 2 == 0:  
        continue # Pula números pares  
    print(numero)
```

#saída:

1  
3  
5  
7  
9

Neste exemplo, o continue ignora os números pares e passa para a próxima iteração, imprimindo apenas os números ímpares.




## ELSE EM LOOPS

Em Python, o else pode ser usado após loops for e while. O bloco else será executado quando o loop terminar normalmente, ou seja, sem que ocorra um break. Isso permite que você trate a conclusão natural do loop de forma diferente do caso em que ele é interrompido prematuramente.

### Aplicações práticas:

- Executar uma ação final apenas quando o loop percorreu todas as iterações.
- Diferenciar se o loop foi interrompido por break ou se terminou de forma natural.

## Exemplo



```
for numero in range(5):  
    print(numero)  
else:  
    print("Loop terminou normalmente.")
```

#saída:

0

1

2

3

4

Loop terminou normalmente.

# PARABÉNS!

Você concluiu a QUARTA  
aula do curso de introdução  
a linguagem PYTHON.

**Agora um desafio:**

Crie um script que imprime  
todos os números ímpares  
entre 1 e 20 usando um loop.

Proxima aula: Listas e funções

