

Loops e Iterações

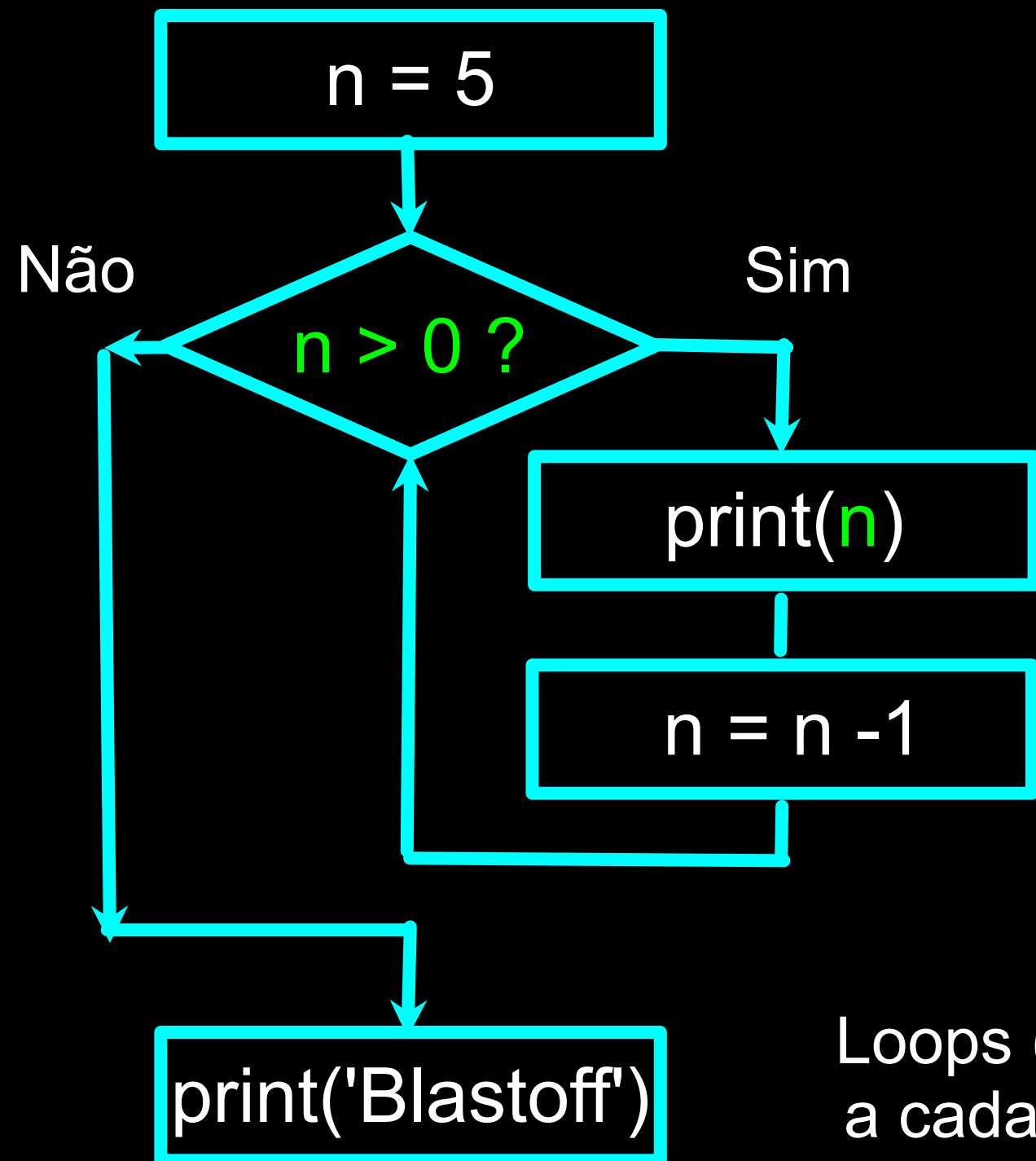
Capítulo 5



Python for Everybody
www.py4e.com



Passos Repetidos



Programa:

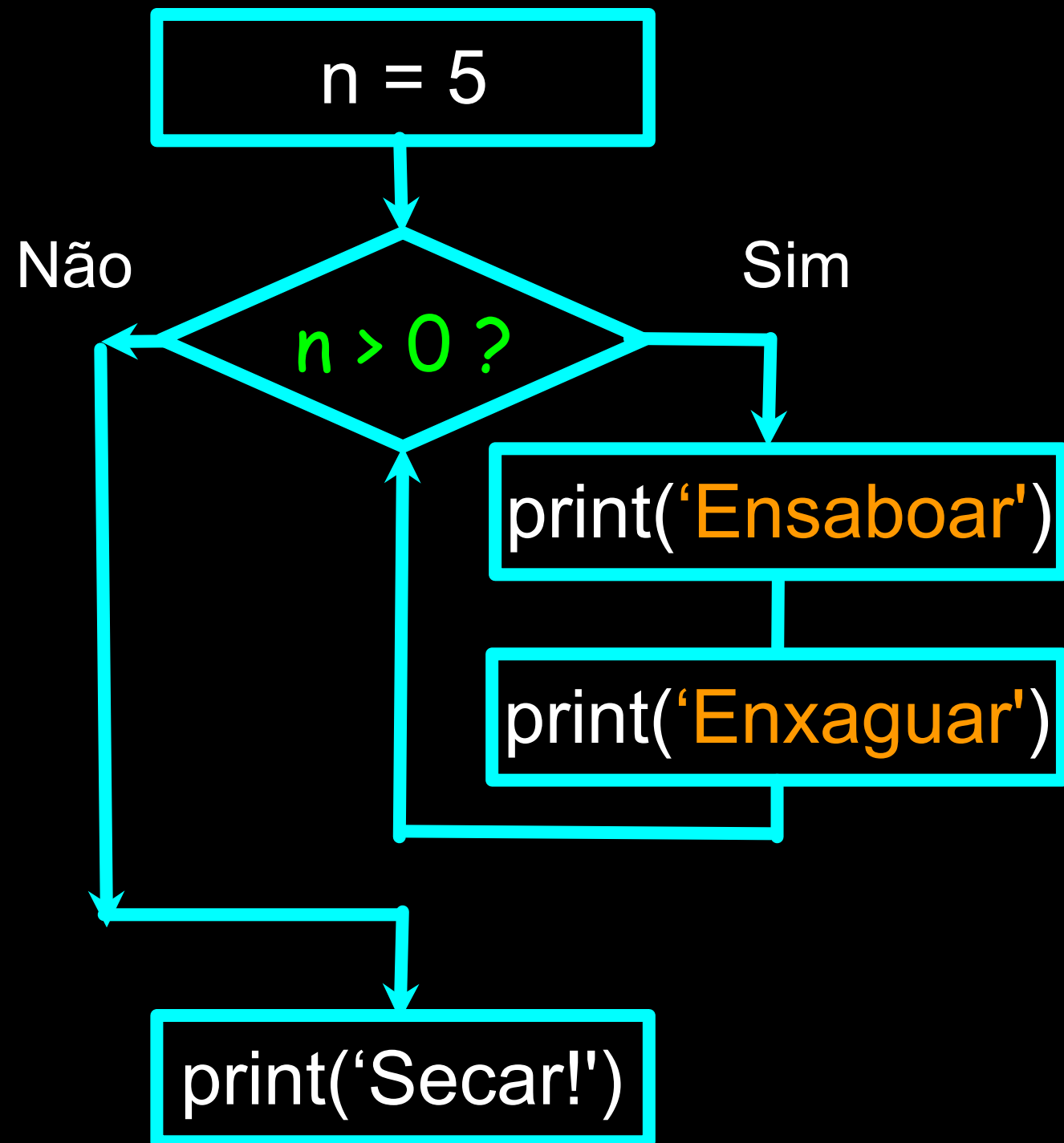
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Decolar!')
print(n)
```

Saída:

5
4
3
2
1
Decolar!
0

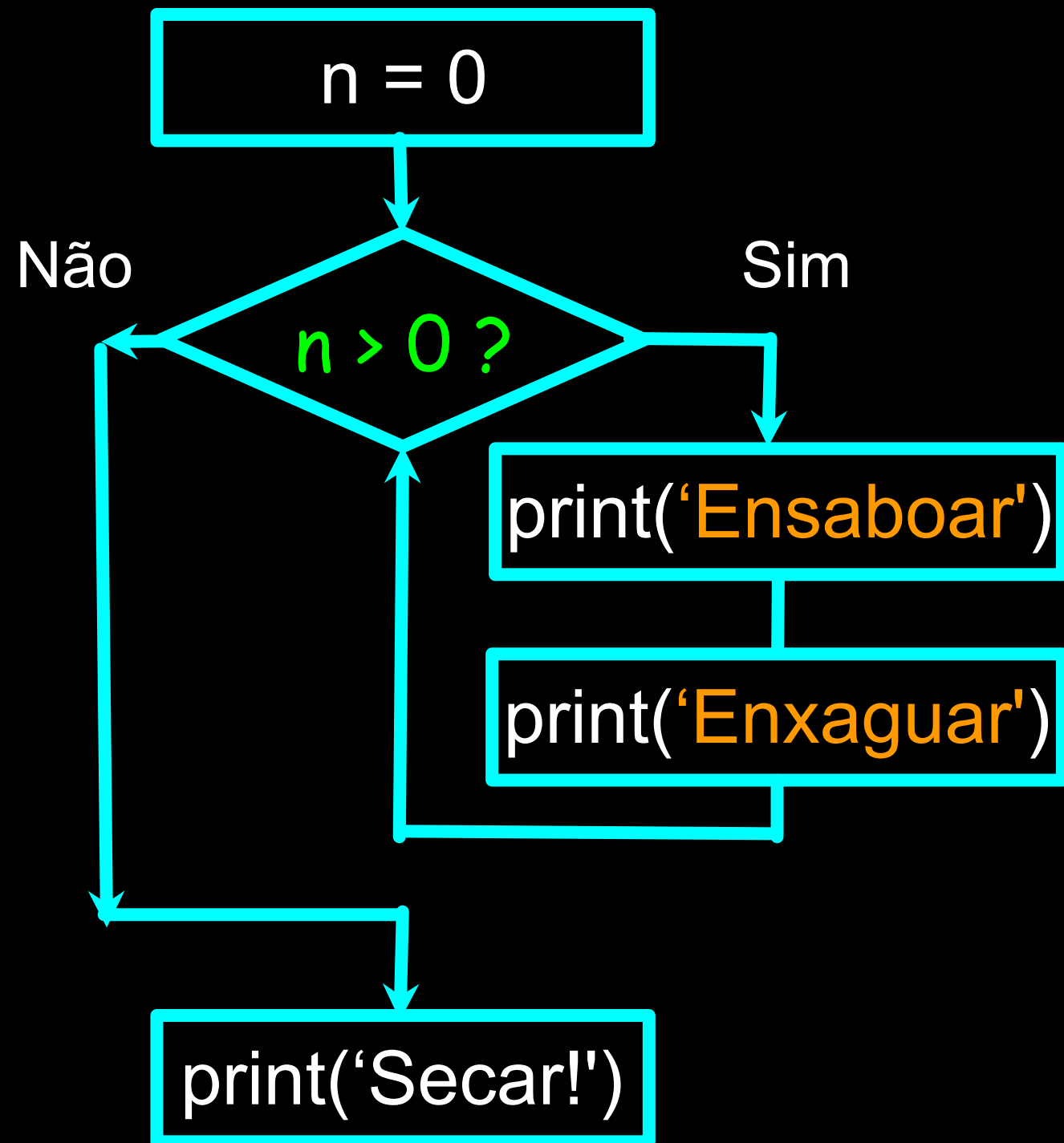
Loops (passos repetidos) têm **variáveis de iteração** que mudam a cada passo que é dado. Frequentemente, essas **variáveis de iteração** passam por uma sequência de números.

Um Loop Infinito



```
n = 5
while n > 0 :
    print('Ensaboar')
    print('Enxaguar')
print('Secar!')
```

O que tem de errado com esse loop?



Outro Loop

```
n = 5
while n > 0 :
    print('Ensaboar')
    print('Enxaguar')
print('Secar!')
```

O que esse loop está fazendo?

Saindo de um Loop com break

- A instrução **break** termina o loop atual e salta para a instrução imediatamente após o loop
- É como um teste de loop que pode acontecer em qualquer parte do corpo do loop


```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

Saindo de um Loop com break

- A instrução **break** termina o loop atual e salta para a instrução imediatamente após o loop
- É como um teste de loop que pode acontecer em qualquer parte do corpo do loop

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

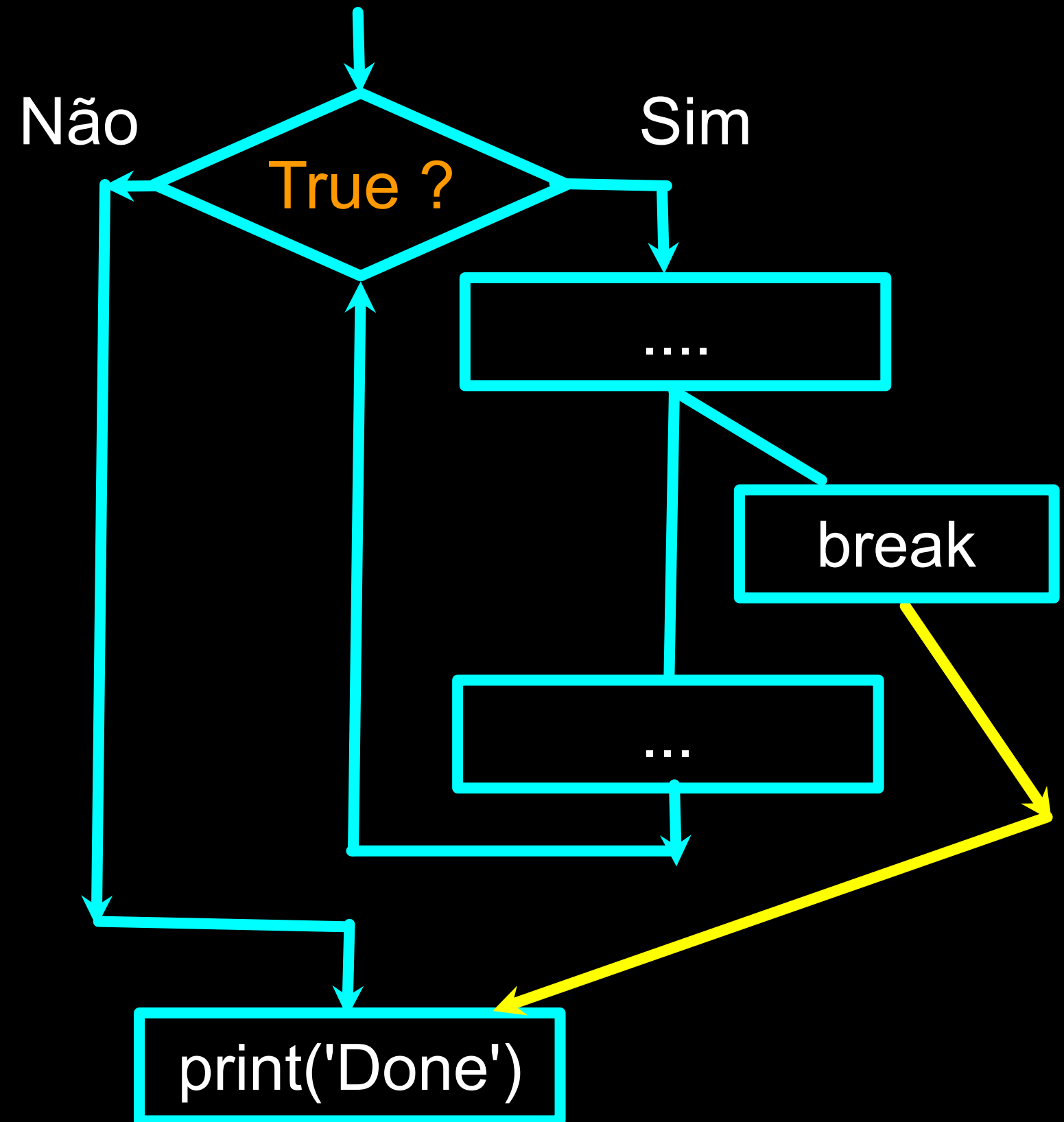


```
> hello there
hello there
> finished
finished
> done
Done!
```

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```



[http://en.wikipedia.org/wiki/Transporter_\(Star_Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



Terminando uma iteração com `continue`

A instrução `continue` termina a iteração atual e salta para a parte superior do loop e inicia a próxima iteração

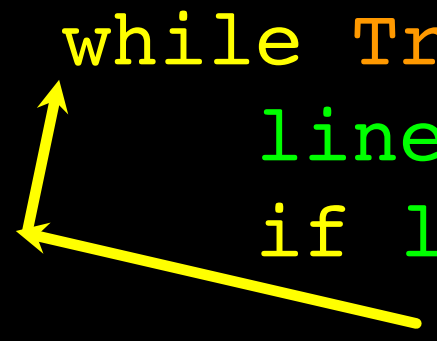
```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```


Terminando uma iteração com `continue`

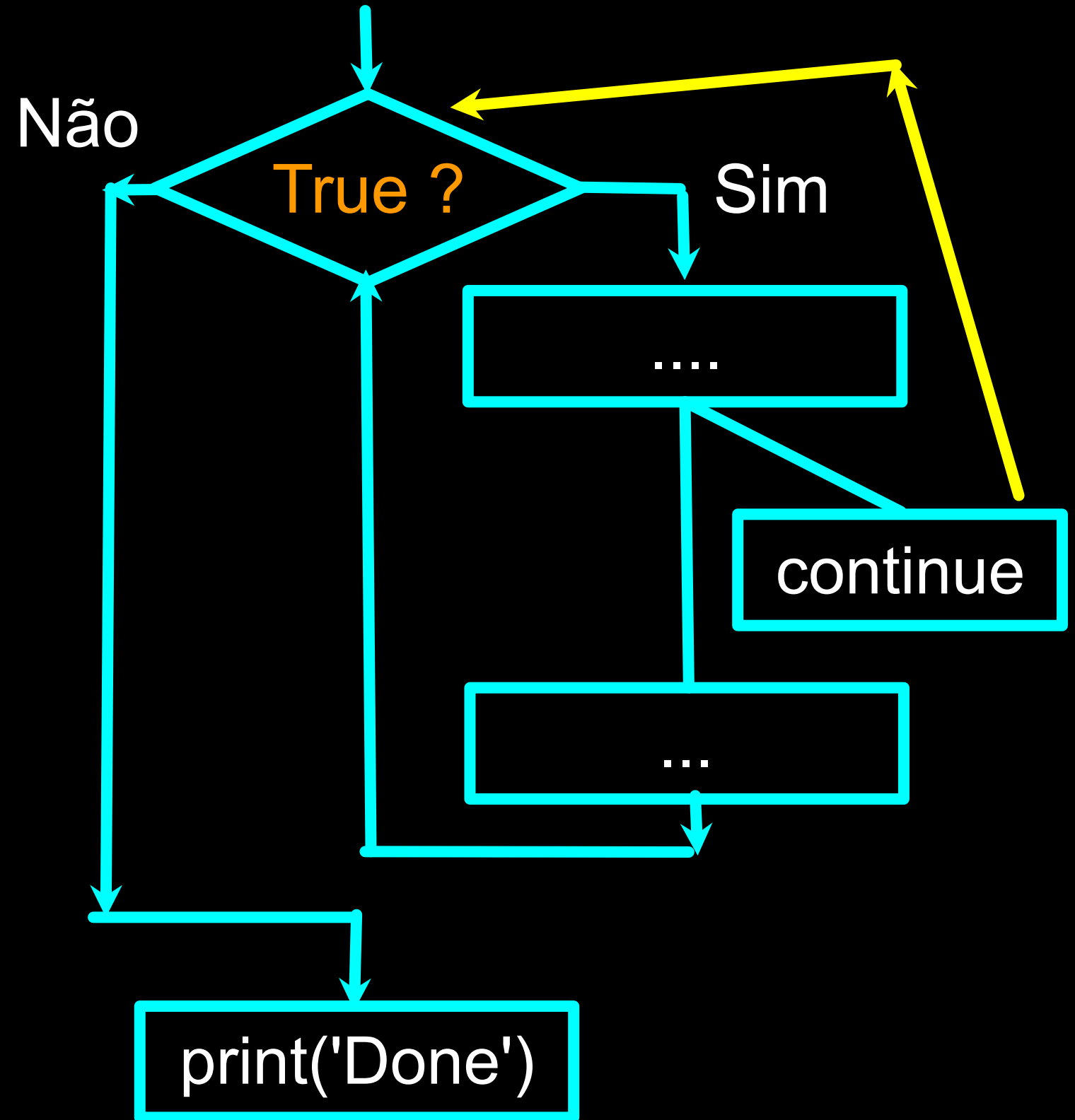
A instrução `continue` termina a iteração atual e salta para a parte superior do loop e inicia a próxima iteração

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```



```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



Loops Indefinidos

- Loops usando **while** são chamados de “**loops indefinidos**” porque continuam até que uma condição lógica se torne False
- Os loops que vimos até agora são bastante fáceis de examinar para ver se eles terminarão ou se serão “loops infinitos”
- Às vezes, é um pouco mais difícil ter certeza se um loop terminará

Loops Definidos

Iterando sobre um conjunto de itens...

Loops Definidos

- Muitas vezes, temos uma **lista** de itens das **linhas em um arquivo**
- efetivamente um **conjunto finito** de coisas
- Podemos escrever um loop para executá-lo uma vez para cada um dos itens de um conjunto usando a instrução **for** em Python
- Esses loops são chamados de “**loops definidos**” porque executam um número exato de vezes
- Dizemos que “**loops definidos iteram pelos elementos de um conjunto**”

Um Loop Definido Simples

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print( 'Decolar!' )
```

5

4

3

2

1

Decolar!

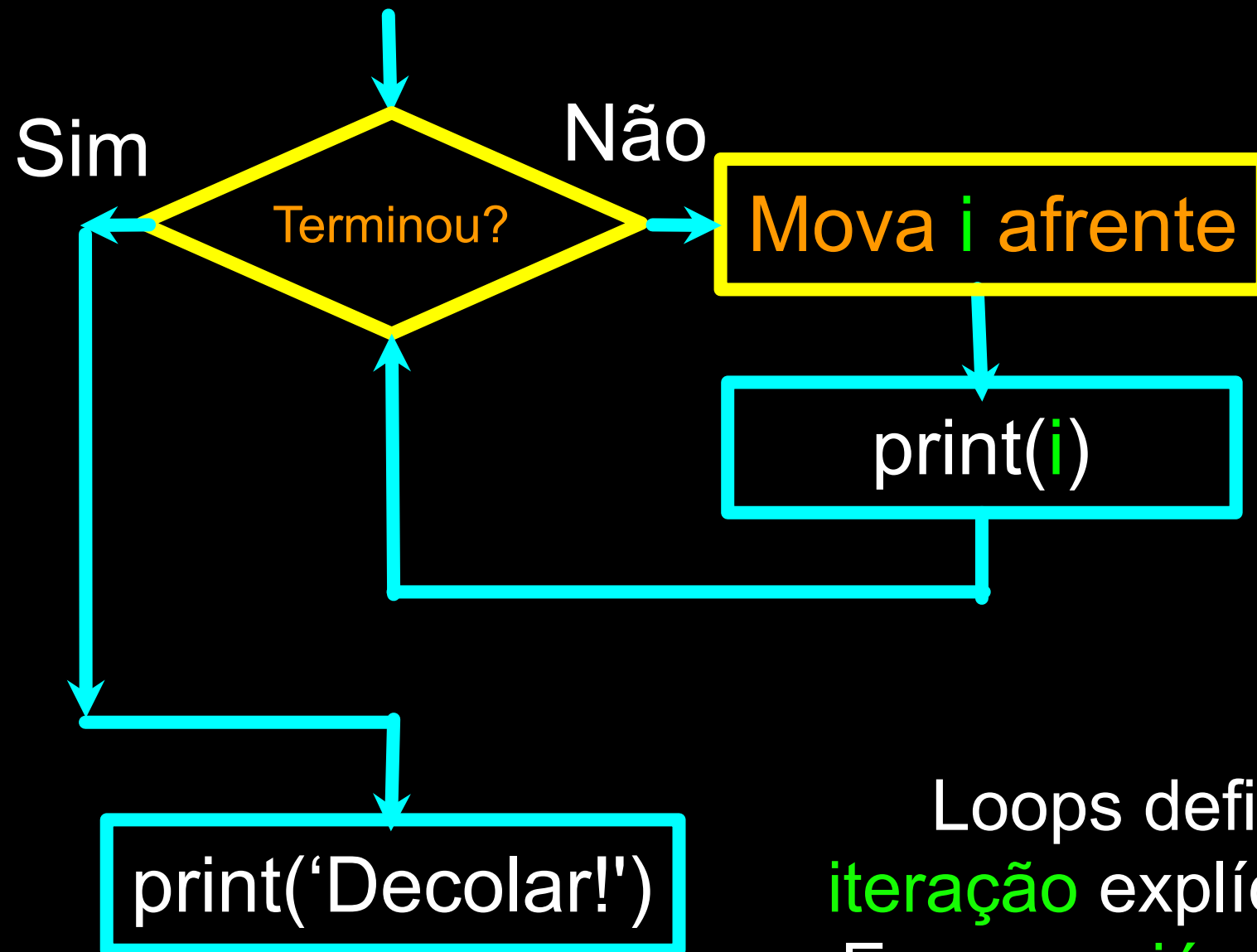
Um Loop Definido com Strings

```
amigos = ['José', 'Glenildo', 'Salu']  
for amigo in amigos :  
    print('Feliz Ano Novo, ', amigo)  
print('Fim!')
```

Feliz Ano Novo, José
Feliz Ano Novo, Glenildo
Feliz Ano Novo, Salu

Fim!

Um Loop Definido Simples



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Decolar!')
```

5
4
3
2
1
Decolar!

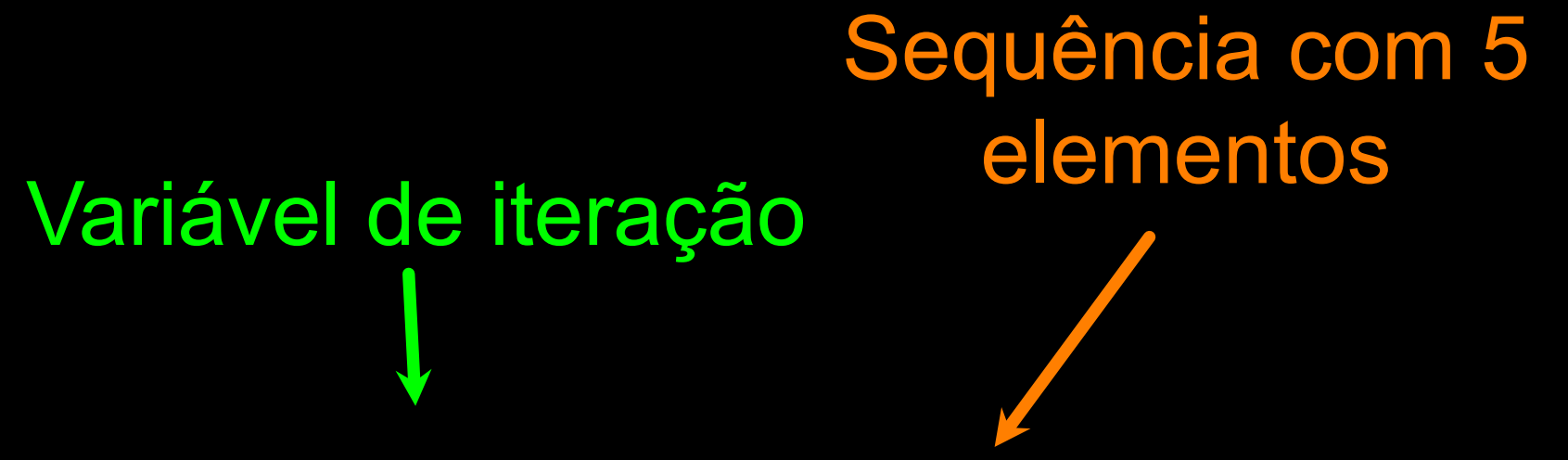
Loops definidos (loops com **for**) têm **variáveis de iteração** explícitas que mudam em cada passo do loop. Essas **variáveis de iteração** se movem pela sequência ou conjunto.

Olhando para o in...

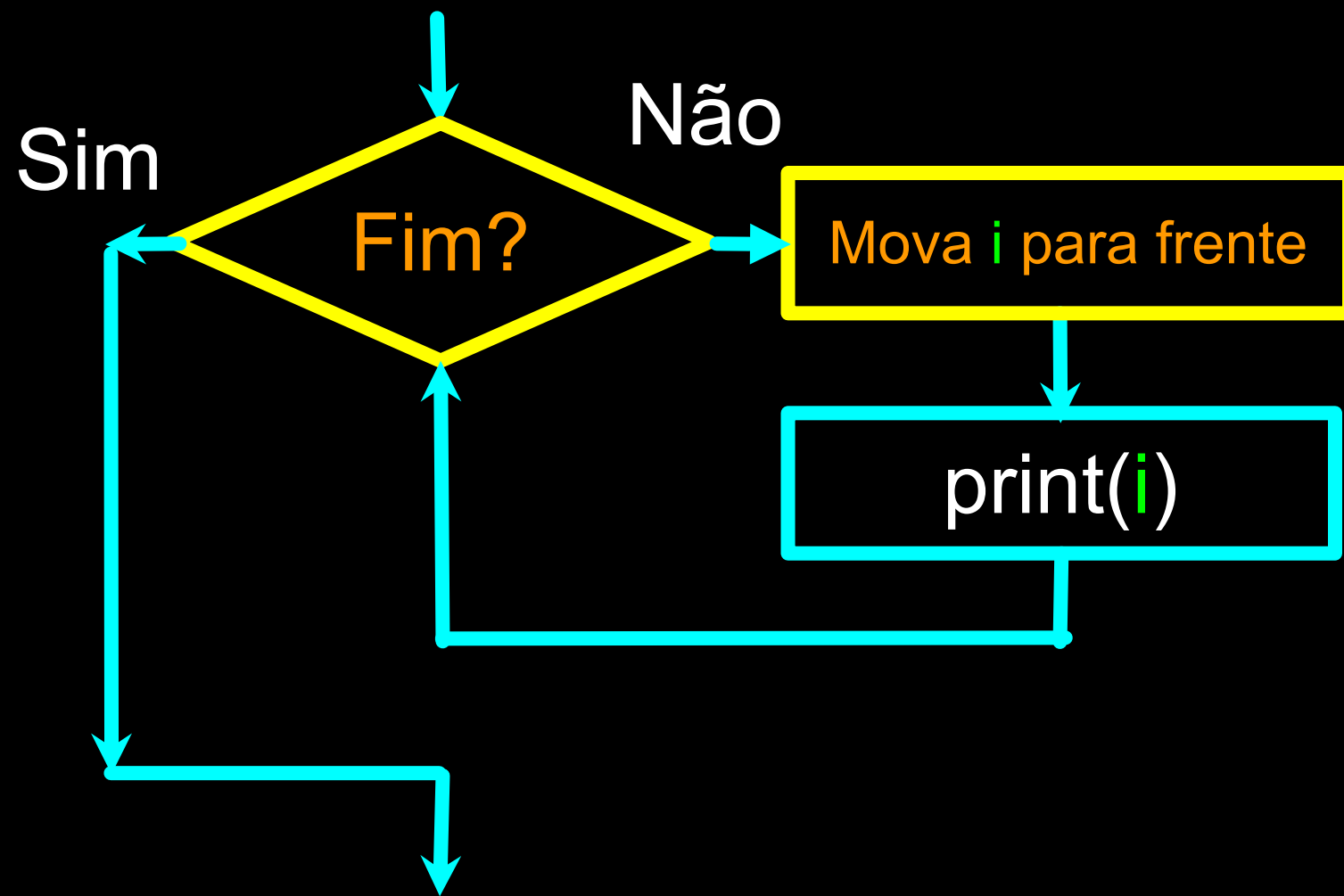
- A **variável de iteração** “itera” pela **sequência** (conjunto ordenado)
- O **bloco (corpo)** do código é executado uma vez para cada valor **na sequência**
- A **variável de iteração** se move por todos os valores **na sequência**

Variável de iteração

Sequência com 5 elementos

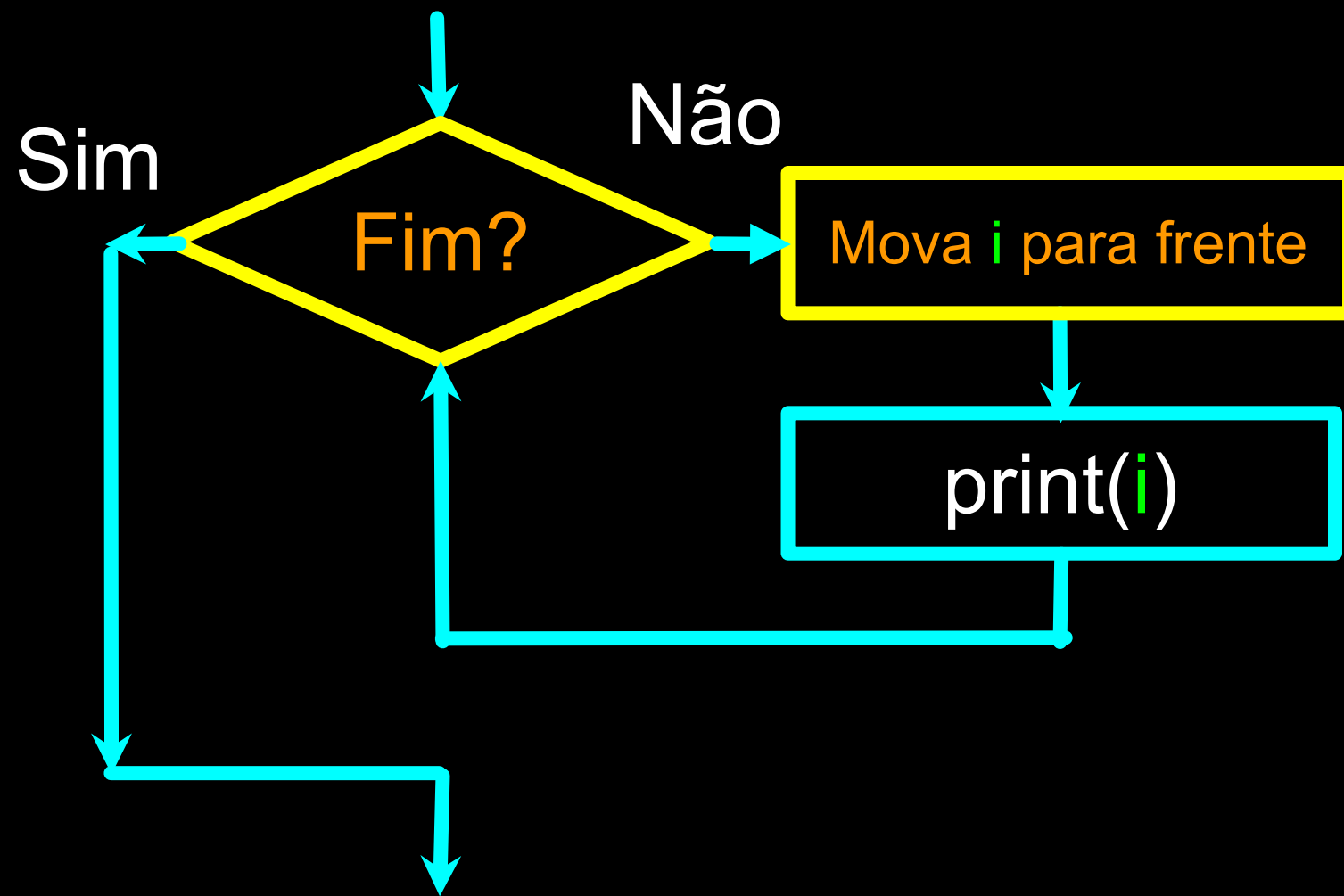


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

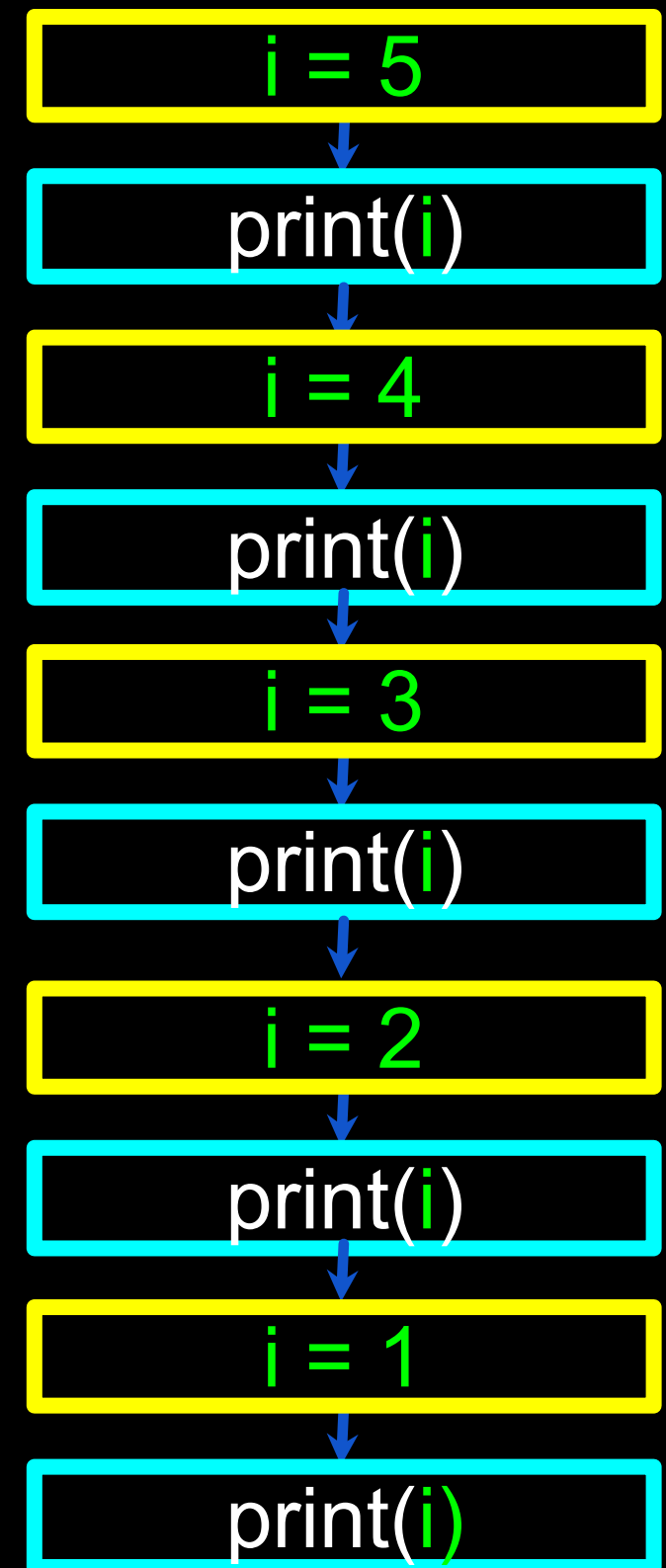


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

- A **variável de iteração** “itera” pela **sequência** (conjunto ordenado)
- O **bloco (corpo)** do código é executado uma vez para cada valor **na sequência**
- A **variável de iteração** se move por todos os valores **na sequência**



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```



Padrões em Loops:

O que fazemos nos Loops

Nota: Embora esses exemplos sejam simples, os padrões se aplicam a todos os tipos de loops

Fazendo loops “inteligentes”

O truque é “saber” algo sobre todo o loop quando você está preso escrevendo código que vê apenas uma entrada de cada vez

Defina algumas variáveis para os valores iniciais

for coisa in dados:

Procure algo ou faça algo para cada entrada separadamente, atualizando uma variável

Veja as variáveis

Loop em um conjunto

```
print('Antes')
for coisa in [9, 41, 12, 3, 74, 15] :
    print(coisa)
print('Depois')
```

\$ python loopbasico.py

Antes

9

41

12

3

74

15

Depois

Qual é o maior número?

Qual é o maior número?

3

Qual é o maior número?

41

Qual é o maior número?

12

Qual é o maior número?

9

Qual é o maior número?

74

Qual é o maior número?

15

Qual é o maior número?

Qual é o maior número?

3

41

12

9

74

15

Qual é o maior número?

maior_ate_agora

-1

Qual é o maior número?

3

maior_ate_agora

3

Qual é o maior número?

41

maior_ate_agora

41

Qual é o maior número?

12

maior_ate_agora

41

Qual é o maior número?

9

maior_ate_agora

41

Qual é o maior número?

74

maior_ate_agora

74

Qual é o maior número?

15

maior_ate_agora

74

Qual é o maior número?

3 41 12 9 74 15

maior_ate_agora

74

Encontrando o Maior Valor

```
maior_ate_agora = -1
print('Antes', maior_ate_agora)
for o_numero in [9, 41, 12, 3, 74, 15] :
    if o_numero > maior_ate_agora :
        maior_ate_agora = o_numero
    print(maior_ate_agora, o_numero)

print('Depois', maior_ate_agora)
```

\$ python maior.py

Antes -1

9 9

41 41

41 12

41 3

74 74

74 15

Depois 74

Definimos uma variável que contém o maior valor visto até agora. Se o **número atual** **que estamos vendo** é maior, ele se torna o **maior valor que vimos até agora**.

Mais padrões de loop...

Contagem em um Loop

```
zork = 0
print('Antes', zork)
for coisa in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, coisa)
print('Depois', zork)
```

```
$ python contadorloop.py
```

```
Antes 0
```

```
1 9
```

```
2 41
```

```
3 12
```

```
4 3
```

```
5 74
```

```
6 15
```

```
Depois 6
```

Para **contar** quantas vezes executamos um loop, introduzimos **uma variável contator que começa com 0** e adicionamos **um toda vez que passarmos no loop**.

Soma em um Loop

```
zork = 0
print('Antes', zork)
for coisa in [9, 41, 12, 3, 74, 15] :
    zork = zork + coisa
    print(zork, coisa)
print('Depois', zork)
```

```
$ python
contadorloop.py
Antes 0
9 9
50 41
62 12
65 3
139 74
154 15
Depois 154
```

Para **adicionarmos** um **valor** que encontramos em um loop, introduzimos **uma variável de soma que começa com 0** e adicionamos **um valor** à soma toda vez que passamos no loop.

Encontrando a média em um Loop

```
contador = 0
sum = 0
print('Antes', contador, sum)
for valor in [9, 41, 12, 3, 74, 15] :
    contador = contador + 1
    sum = sum + valor
    print(contador, sum, valor)
print('Depois', contador, sum, sum / contador)
```

\$ python loopmedia.py

Antes 0 0

1 9 9

2 50 41

3 62 12

4 65 3

5 139 74

6 154 15

Depois 6 154 25.666

A **média** apenas combina os padrões de **contagem** e **soma** e divide quando o loop termina.

Filtrando em um Loop

```
print('Antes')
for valor in [9, 41, 12, 3, 74, 15] :
    if valor > 20:
        print('Número grande',valor)
print('Depois')
```

```
$ python busca1.py
Antes
Número grande 41
Número grande 74
Depois
```

Usamos uma declaração **if** no **loop** para capturar / filtrar os valores que estamos procurando.

Busca usando uma variável booleana

```
found = False
print('Antes', found)
for valor in [9, 41, 12, 3, 74, 15] :
    if valor == 3 :
        found = True
    print(found, valor)
print('Depois', found)
```

```
$ python search1.py
Antes False
False 9
False 41
False 12
True 3
True 74
True 15
Depois True
```

Se quisermos apenas buscar e **e saber se o valor foi encontrado**, usamos uma **variável** que começa **False** e que é setada para **True** assim que **encontramos** o que estamos procurando.

Como encontrar o menor valor

```
maior_ate_agora = -1
print('Antes', maior_ate_agora)
for o_numero in [9, 41, 12, 3, 74, 15] :
    if o_numero > maior_ate_agora :
        maior_ate_agora = o_numero
    print(maior_ate_agora, o_numero)

print('Depois', maior_ate_agora)
```

```
$ python maior.py
```

```
Antes -1
```

```
9 9
```

```
41 41
```

```
41 12
```

```
41 3
```

```
74 74
```

```
74 15
```

```
Depois 74
```

Como podemos mudar esse código para encontrar o menor valor na lista?

Encontrando o Menor Valor

```
menor_ate_agora = -1
print('Antes', menor_ate_agora)
for o_numero in [9, 41, 12, 3, 74, 15] :
    if o_numero < menor_ate_agora :
        menor_ate_agora = o_numero
    print(menor_ate_agora, o_numero)

print('Depois', menor_ate_agora)
```

Mudamos o nome da variável para `menor_ate_agora` e trocamos o `>` por `<`

Encontrando o Menor Valor

```
menor_ate_agora = -1
print('Antes', menor_ate_agora)
for o_numero in [9, 41, 12, 3, 74, 15] :
    if o_numero < menor_ate_agora :
        menor_ate_agora = o_numero
    print(menor_ate_agora, o_numero)

print('Depois', menor_ate_agora)
```

\$ python menorrui.py

Antes -1

-1 9

-1 41

-1 12

-1 3

-1 74

-1 15

Depois -1

Mudamos o nome da variável para `menor_ate_agora` e trocamos o `>` por `<`

Encontrando o Menor Valor

```
menor = None
print('Antes')
for valor in [9, 41, 12, 3, 74, 15] :
    if menor is None :
        menor = valor
    elif valor < menor :
        menor = valor
    print(menor, valor)
print('Depois', menor)
```

\$ python menor.py

Antes

9 9

9 41

9 12

3 3

3 74

3 15

Depois 3

Ainda temos uma variável que é **a menor** até agora. A primeira vez que entramos no loop **a variável menor** é **None**, então nós pegamos o primeiro **valor** para ser o **menor**.

Os Operadores **is** e **is not**

```
menor = None
print('Antes')
for valor in [3, 41, 12, 9, 74, 15] :
    if menor is None :
        menor = valor
    elif valor < menor :
        menor = valor
    print(menor, valor)

print('Depois', menor)
```

- Python tem um operador **is** que pode ser usado em expressões lógicas
- Indica “**is the same as**” (“é o mesmo que”)
- Similar, mas mais forte que **==**
- **is not** também é um operador lógico

Resumo

- Loops com while (indefinidos)
- Loops infinitos
- Usando break
- Usando continue
- Usando None
- Loops com for (definidos)
- Variáveis de Iteração
- Padrões de Loops
- Maior ou menor



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Traduzido para o Português Brasileiro por Filipe Calegario, Centro de Informática, Universidade Federal de Pernambuco

...