IBGE
Instituto Brasileiro de Geografia e Estatística

Escola Nacional de Ciencias Estat sticas

Introdução à Programação

Aula 08: Repetição (parte 2): instrução for; instruções break e continue; laços aninhados

Prof. Eduardo Corrêa

Data 02/04/2024

Tópicos da Aula

- Temas desta aula:
  - A Estrutura de repetição **for-range()**
  - Quando usar o for? Quando usar while?
  - Instruções break e continue
  - Laços aninhados

O comando for (1/2)

- Assim como o **while**, o **for** é um comando de repetição.
  - Ele existe em quase todas as linguagens de programação.
  - Porém é mais simples
  - Serve para definir a execução de um laço por um **número fixo** de iterações (repetições).
  - **Ex.**: laço com 5000 iterações.

```
for i in range(5000):
    comando1
    comando2
    …
    comandom
```

O Comando for (2/2)

- Na linguagem Python o **for** pode iterar apenas sobre **coleções**
  - "iterar" significa realizar as repetições
  - "coleção" significa uma estrutura que contém um conjunto de valores onde o for vai percorrer.
- Nessa aula, mostraremos como implementar um **for básico**, usando a função **range()** para gerar as coleções.

for-range() básico (1/5)

- O código abaixo ilustra o modelo mais simples de utilização do **for-range()** (existem outros, mas começaremos com esse):

```
for v in range(n):
    comando1
    comando2
    …
    comandom
```

Estrutura for-range()

```
for v in range(n):
    comando_1
    comando_2
    ...
    comando_m
```

- "v" é o nome da variável de controle. Essa variável será tratada como uma variável inteira.

for-range() básico (2/5)

```
for v in range(n):
    comando_1
    comando_2
    …
    comando_m
```

- "n" corresponde ao tamanho da sequência que desejamos gerar.
- Quando utilizamos **range(n)**, o resultado será uma sequência de números inteiros que começa com 0 e vai até n-1: {0, 1, 2, ..., n-1}

for-range() básico (3/5)

```
for v in range(n):
    comando1
    comando2
    …
    comandom
```

- Durante a execução do laço, a variável de controle assume o valor inicial da sequência na primeira iteração (neste caso, o valor 0).
- A cada iteração, o valor da variável de controle será **automaticamente modificado** para o próximo valor da sequência.
- O laço termina, quando o seu valor atingir o valor final, ou seja, **n-1**.

for-range() básico (4/5)

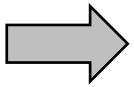
```
for v in range(n):
    comando1
    comando2
    …
    comandom
```

- Em resumo:
  - o **for** utilizado em conjunto com **range(n)** realizará exatamente **n iterações**.
  - A cada iteração, a variável de controle assumirá um dos valores da sequência, começando do primeiro e terminando no último.

for-range() básico (5/5)

- **Exemplo**: Imprimir 15 linhas.

```
for i in range(15):
    print('LINHA ',i)
```



```
>>> %Run im
LINHA  0
LINHA  1
LINHA  2
LINHA  3
LINHA  4
LINHA  5
LINHA  6
LINHA  7
LINHA  8
LINHA  9
LINHA  10
LINHA  11
LINHA  12
LINHA  13
LINHA  14
```

- O programa que usa um laço montado com **for-range()** para imprimir 15 vezes a palavra LINHA, seguida do valor da variável "i" (variável de controle do for).
- Observe que o valor de "i" muda (aumenta em 1) sozinho para cada iteração do loop, uma vez que range(15) gera a sequência {0, 1, 2, ..., 14}.

Detalhando a função range() (1/4)

- A função **range()** serve para gerar uma sequência de números inteiros.
  - A sequência é geralmente utilizada para ser percorrida por um comando **for**.
- Acabamos de ver que range(n) serve para gerar a sequência {0, 1, 2, ..., n-1}.
  - Esta sequência pode ser utilizada para definir um laço com n iterações, em que a variável de controle assumirá o valor 0 na **primeira iteração** e o valor *n*-1 **na última**.

Detalhando a função range() (2/4)

- Entretanto, a função range() é sofisticada !!!
- Ela possibilita a geração de outros tipos de sequência de números inteiros.
  - **Por exemplo**: sequências que começam com outro valor diferente de 0 e até mesmo sequências decrescentes.
  - Vamos aprender a fazer isso?
    - O segredo é entender a sintaxe completa da função range(), apresentada no slide a seguir.

Detalhando a função range() (3/4)

- **range()** aceita que até 3 parâmetros sejam utilizados para definir uma sequência: início, fim e incremento.

• **Sintaxe: range**(início, fim, incremento)

• **início:** número inicial da sequência (opcional). Caso seja omitido, o valor 0 é assumido.

• **fim:** a sequência será gerada até, mas sem incluir, o número especificado neste parâmetro (único parâmetro obrigatório). ENTÃO MUITO CUIDADO!!! A sequência **nunca** irá incluir o valor que você especificar no parâmetro fim. Ela sempre vai parar 1 valor antes.

• **incremento:** diferença entre cada número na sequência (opcional). Se omitido, o valor 1 é adotado.

• **range**(**3**) *# {0, 1, 2}*
• **range**(**1, 4**) *# {1, 2, 3}*
• **range**(**0, 10, 2**) *# {0, 2, 4, 6, 8}*

Detalhando a função range() (4/4)

```
print('\n* * seq. de 0 a 3')
for i in range(4): print(i)

print('\n* * seq. de 10 a 15')
for i in range(10, 16):
  print(i)

print('\n* * ordem reversa:')
for i in range(3, 0, -1):
  print(i)
```

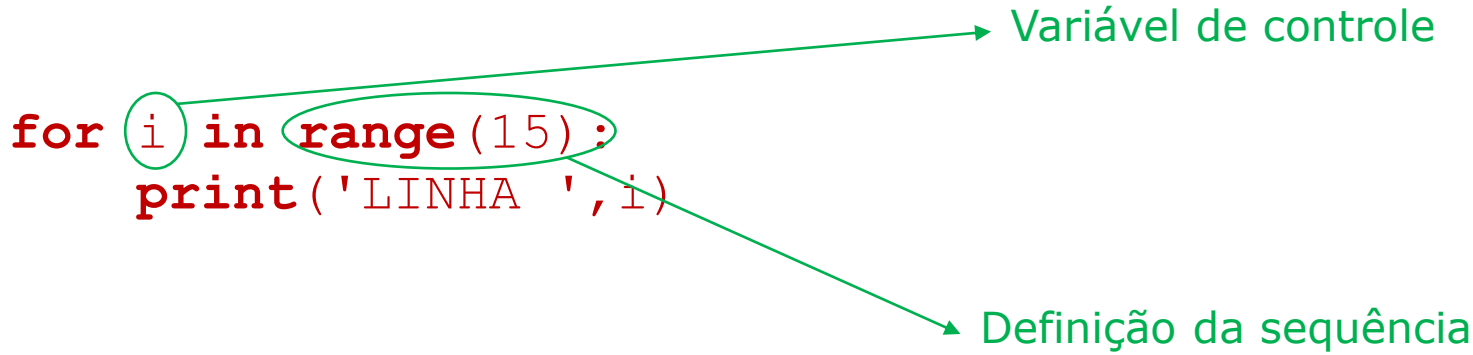
```
>>>
* * seq. de 0 a 3
0
1
2
3

* * seq. de 10 a 15
10
11
12
13
14
15

* * ordem reversa:
3
2
1
```

Considerações importantes (1/5)

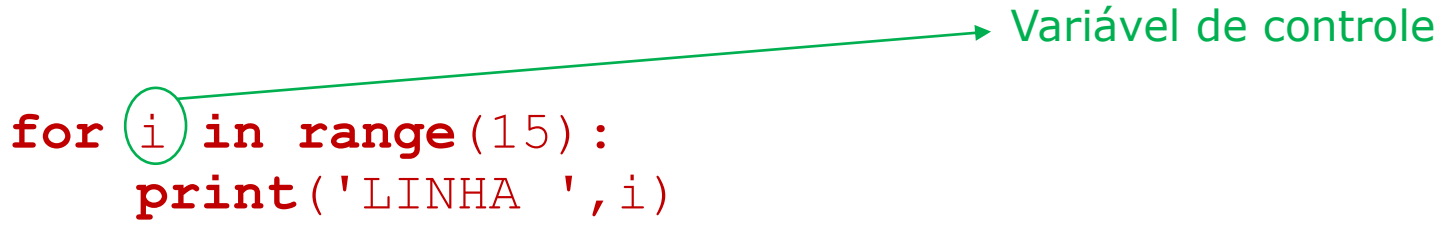
- **Características do for-range()**



- O laço é executado por um **número fixo de vezes**, bastando para isso especificar 2 coisas:
  - Uma variável de controle.
  - Uma sequência com a função range().

Considerações importantes (2/5)

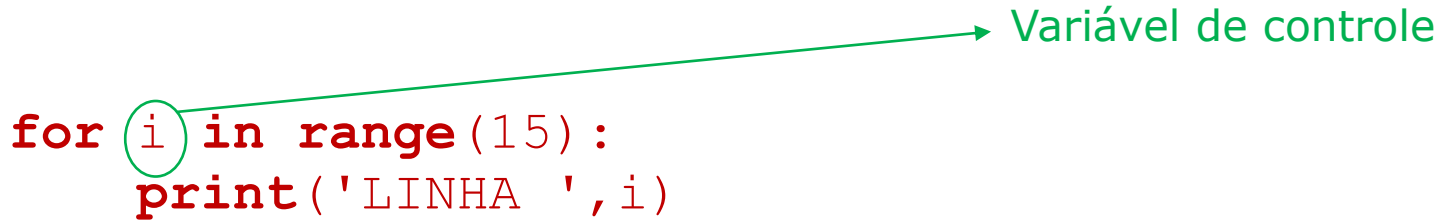
- **Características do for-do**



- O programador **não tem** que mexer no valor da variável de controle dentro do laço!
  - Ele é **incrementado automaticamente** ao final de cada iteração do loop.
  - Na 1ª iteração do loop o valor da variável de controle será igual ao primeiro da sequência gerada pelo range().
  - Na 2ª será igual ao segundo valor da sequência.
  - E na última, o valor será igual ao do último da sequência.

Considerações importantes (3/5)

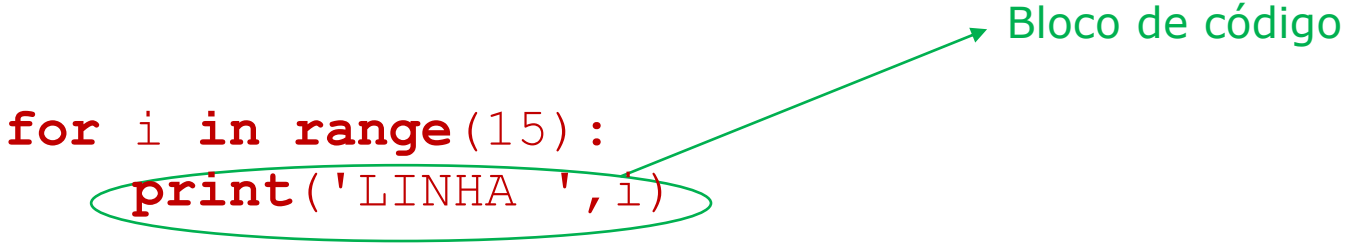
- **Características do for-do**



- Não é preciso atribuir nenhum valor para a variável controladora **antes** da execução do comando for
  - Ao contrário do que ocorre com o comando while...

Considerações importantes (4/5)

- **Características do for-do**



- Os bloco de código subordinado ao for deve estar indentado, abaixo do for

Obs: *se houver **um único comando** pode ficar ao lado, como ocorre no if e elif. Então, no exemplo acima seria possível colocar ao lado*

Considerações importantes (5/5)

- **Características do for-do**

```
for i in range(15):
    print('LINHA ',i)
```

- Com o uso do **for** o programa **nunca entra em loop infinito**, pois ele será repetido por um número fixo de vezes.
- O programa só corre risco de entrar em loop infinito se o programador alterar de forma equivocada o valor da variável de controle dentro dos comandos subordinados ao for.
  - Embora isso seja permitido, em geral, não é recomendável.

for *versus* while

- Tudo que fazemos com o **for**, podemos implementar com o **while**. Mas a recíproca não é verdadeira!

- Se você vai executar um laço por um número previsível (**fixo**) de vezes, pode usar o for. **Exemplos**:
  - Programa que recebe a nota de 50 alunos e calcula a média.
  - Programa que determina se um número N é primo (basta executar até a raiz de N).

- Mas se o número de repetições é **imprevisível**, não há como usar o for.
  - **Exemplo**: Programa mantido em execução até o usuário digitar o valor -1 (não sei de antemão quando ele vai fazer isso!)

for Reverso

- Conforme visto, é possível definir uma sequência decrescente na função range() .
- Basta usar um "incremento" negativo e definir o "fim" com um valor maior do que o do "início".

```
for i in range(5, 0, -1):  print(i)
```

```
>>>
5
4
3
2
1
```

- Uma forma ainda mais simples de fazer o for "reverso" é utilizando a função reversed().
- É simples: você especifica uma sequência normal e a função reversed() a inverte pra você!

```
for i in reversed(range(1,6)):
  print(i)
```

```
>>>
5
4
3
2
1
```

break e continue (1/4)

- break e continue são dois comandos muito úteis na implementação de laços.
- Ambos podem ser utilizados dentro do while ou dentro do for.
  - **break**: pode ser utilizado **quebrar um laço** "na marra", passando o fluxo de execução do programa para a linha que estiver localizada imediatamente depois do fim do bloco de comandos subordinados ao laço.
  - **continue**: serve para **quebrar uma iteração**, mas não o laço propriamente dito. Mais claramente: sempre que o continue é executado, o fluxo de execução do programa é automaticamente desviado para a linha que contém o comando while ou for.

break e continue (2/4)

```
print('(1)-Exemplo - break:')
n = -1
while (n < 21):
    n += 1
    if n % 2 != 0:
        break #quebra o laço se n for ímpar...
    print(n)

print('fim...')
```

Saída:

```
(1)-Exemplo - break:
0
fim...
```

break e continue (3/4)

```
print('(1)-Exemplo - continue:')
n = -1
while (n < 21):
    n += 1
    if n % 2 != 0:
        continue # quebra iteração
                 # se n for ímpar
                 # mas o laço não
    print(n)

print('fim...')
```

Saída:

```
(2)-Exemplo - continue:
0
2
4
6
8
10
12
14
16
18
20
fim...
```

break e continue (4/4)

- O comando **break** é particularmente útil para combinar com um **while** True.
- Os dois juntos definem um loop só será quebrado pelo break. Veja o exemplo abaixo: um programa vai somando números até o usuário informar que quer sair.

```
soma = 0
while True:
    soma += float(input('Digite um numero: '))
    opcao = input('Digite "S" p/ sair ou ENTER p/ continuar: ')
    if opcao == 'S': break

print(f'tudo somado deu: {soma:.2f}')
```

Saída:

```
Digite um numero: 1.55
Digite "S" para sair ou ENTER para continuar:
Digite um numero: 3.6
Digite "S" para sair ou ENTER para continuar:
Digite um numero: 5.4
Digite "S" para sair ou ENTER para continuar: S
tudo somado deu: 10.55
```

Laços Aninhados (1/2)

- Muitos problemas só podem ser resolvidos com a utilização de **loops aninhados**,
  - Isto significa um laço dentro de outro laço.
  - Exemplos:
    - Utilizar um for dentro de outro for
    - um while dentro de um for
    - um while dentro de outro while
    - etc.
  - Em muitos casos, é preciso até mesmo utilizar mais de 2 laços aninhados
    - Ex.: while dentro de um for dentro de outro for.

Laços Aninhados (2/2)

- **Exemplo**: Tabuada de 1 a 10.

```
for i in range(1,11):
    print('Tabuada de',i)
    for j in range(1,11):
        print(i,'x',j,'=',i*j)
    print()
```

```
Tabuada de 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10

Tabuada de 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

Tabuada de 3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
```

Exercícios

- (1) Utilizando for, escreva um programa que leia a nota final de 10 alunos de uma turma e que calcule e imprima a média da turma.

- (2) Utilizando o comando for, construir um programa Python que leia um número inteiro positivo n como entrada e calcule e exiba na tela o valor de n! (ou seja, o fatorial de n).

- (3) O número 3025 possui a seguinte característica:
  $30 + 25 = 55$
  $55^2 = 3025$

  Crie um programa que pesquise e imprima todos os números de quatro algarismos que apresentem tais características. (*tem que usar o for também!*)