Programação Estruturada Linguagem Python

Professor Adjenor Cristiano Queiroz
FAPAM

Aula 7

Estrutura For

Python apresenta uma estrutura de repetição especialmente projetada para percorrer listas. A instrução for funciona de forma parecida a while, mas a cada repetição utiliza um elemento diferente da lista.

A cada repetição, o próximo elemento da lista é utilizado, o que se repete até o fim da lista. Vamos escrever um programa que utilize for para imprimir todos os elementos de uma lista (Listagem 6.24).

▶ Listagem 6.24 – Impressão de todos os elementos da lista com for

```
L=[8,9,15]
for e in L: ❶
    print(e) ❷
```

Quando começamos a executar o for em ❶, temos e igual ao primeiro elemento da lista, no caso, 8, ou seja, L[0]. Em ❷ imprimimos 8, e a execução do programa volta para ❶, onde e passa a valer 9, ou seja, L[1]. Na próxima repetição e valerá 15, ou seja, L[2]. Depois de imprimir o último número, a repetição é concluída, pois não temos mais elementos a substituir.

Se tivéssemos que fazer a mesma tarefa com while, teríamos que escrever um programa como o da listagem 6.25

▶ Listagem 6.25 – Impressão de todos os elementos da lista com while

```
L=[8,9,15]
x=0
while x<len(L):
    e=L[x]
    print(e)
    x+=1
```

Embora a instrução for facilite nosso trabalho, ela não substitui completamente while. Dependendo do problema, utilizaremos for ou while. Normalmente utilizaremos for quando quisermos processar os elementos de uma lista, um a um. While é indicado para repetições nas quais não sabemos ainda quantas vezes vamos repetir ou onde manipulamos os índices de forma não sequencial.

Vale lembrar que a instrução break também interrompe o for. Vejamos a pesquisa, escrita usando for, na listagem 6.26.

▶ **Listagem 6.26 – Pesquisa usando for**

```
L=[7,9,10,12]
p=int(input("Digite um número a pesquisar:"))
for e in L:
    if e == p:
        print("Elemento encontrado!")
        break ❶
else: ❷
    print("Elemento não encontrado.")
```

Utilizamos a instrução break para interromper a busca depois de encontrarmos o primeiro elemento em ❶. Em ❷ utilizamos um else, parecido com o da instrução if, para imprimir a mensagem informando que o elemento não foi encontrado.

O else deve ser escrito na mesma coluna de for e só será executado se todos os elementos da lista forem visitados, ou seja, se não utilizarmos a instrução break, deixando for terminar normalmente.

Range:

Podemos utilizar a função range para gerar listas simples. A função range não retorna uma lista propriamente dita, mas um gerador ou generator. Por enquanto, basta entender como podemos usá-la. Imagine um programa simples que imprime de 0 a 9 na tela (Listagem 6.27).

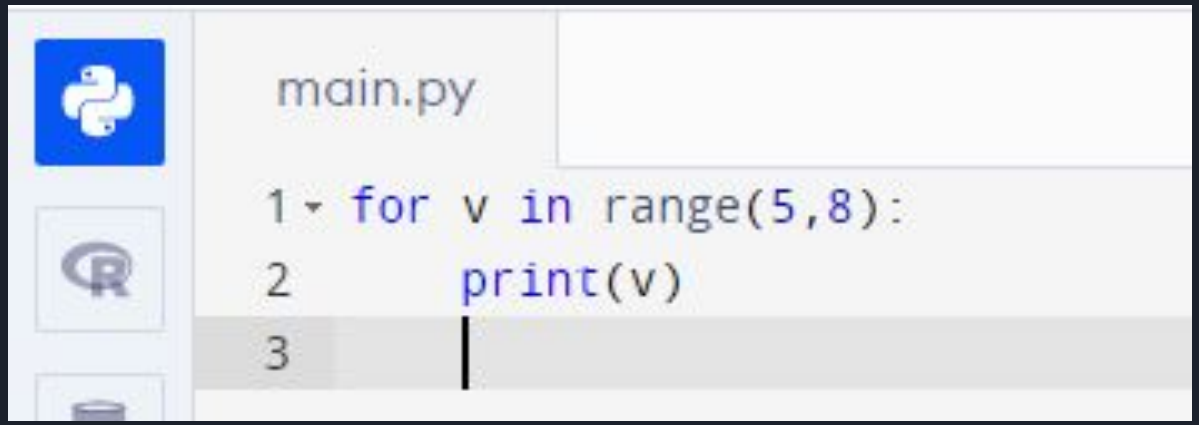
▶ Listagem 6.27 – Uso da função range

```
for v in range(10):
    print(v)
```

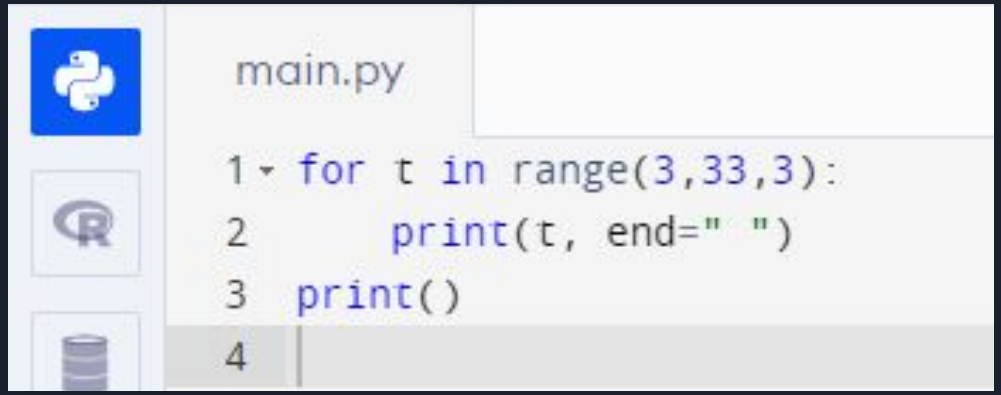
Range:

https://www.programiz.com/python-programming/online-compiler/

A vantagem de utilizarmos a função range é gerar listas eficientemente, como mostrado no exemplo, sem precisar escrever os 20.000 valores no programa.
Com a mesma função range, podemos também indicar qual é o primeiro número a gerar. Para isso, utilizaremos dois parâmetros: início e fim:



Se acrescentarmos um terceiro parâmetro à função range, teremos como saltar entre os valores gerados, por exemplo, range(0,10,2) gera os pares entre 0 e 10, pois começa de 0 e adiciona 2 a cada elemento. Vejamos um exemplo onde geramos os 10 primeiros múltiplos de 3



Observe que um gerador como o retornado pela função range não é exatamente uma lista. Embora seja usado de forma parecida, é, na realidade, um objeto de outro tipo. Para transformar um gerador em lista, utilize a função list

main.py

```
1 L=list(range(100,200,10))
2 print(L)
```

Shell

```
[100, 110, 120, 130, 140, 150, 160, 170, 180, 190]
```

Enumerate

Com a função enumerate podemos ampliar as funcionalidades de for facilmente.

Vejamos como imprimir uma lista, onde teremos o índice entre colchetes e o valor à sua direita

```
>>> Lista = [4,9,1,8]
>>> for cont, item in enumerate(Lista):
        print("Item: ",item, " | Posição: ",cont)


Item:  4  | Posição:  0
Item:  9  | Posição:  1
Item:  1  | Posição:  2
Item:  8  | Posição:  3
```

A função enumerate gera uma tupla em que o primeiro valor é o índice e o segundo é o elemento da lista sendo enumerada.

Verificação do maior valor:

```
>>> Lista = [4,9,1,8]
>>> maior = 0
>>> for cont, item in enumerate(Lista):
        if((item > maior)or(cont==0)):
                maior=item


>>> print(maior)
9
>>>
```

Exercícios:

1 - Altere o programa anterior de modo que ele receba os valores para preencher a lista, a cada valor digitado o sistema ofereça ao usuário a opção de imprimir a lista com seus índices ([0] - 4). O programa também deve oferecer ao usuário a opção de adicionar mais um valor ou excluir um valor. Ao finalizar o sistema deve apresentar na tela **todos os números digitados, o maior, menor e a média.**

Exercícios:

2 - Desenvolva um programa para separar os valores da lista [9,8,7,12,0,13,21,35,6,11,1] em duas listas, uma com os pares e outra com os ímpares.

Listas dentro de Listas

Array Multidimensional

Um fator interessante é que podemos acessar as strings dentro da lista, letra por letra, usando um segundo índice.

```
>>> S=["maçãs", "peras", "kiwis"]
>>> print(S[0][0])
m
>>> print(S[0][1])
a
>>> print(S[1][1])
e
>>> print(S[2][2])
w
```

▶ Listagem 6.39 – Impressão de uma lista de strings, letra a letra

```
L=["maçãs", "peras", "kiwis"]
for s in L:
    for letra in s:
        print(letra)
```

Isso nos mostra as vantagens de se trabalhar com listas em Python. Vale ressaltar que os elementos de uma lista não precisam ser do mesmo tipo.

Vejamos um exemplo onde teríamos uma lista de compras. O primeiro elemento seria o nome do produto; o segundo, a quantidade e o terceiro seu preço.

▶ Listagem 6.40 – Listas com elementos de tipos diferentes

```
produto1 = [ "maçã", 10, 0.30]
produto2 = [ "pera",  5, 0.75]
produto3 = [ "kiwi",  4, 0.98]
```

Assim, produto1, produto2, produto3 seriam três listas com três elementos cada uma.

Observe que o primeiro elemento é do tipo string; o segundo, do tipo inteiro; e o terceiro, do tipo ponto flutuante (float)!

Também podemos adicionar Listas em outras listas e acessar os valores separadamente:

```
produto1 = [ "maçã", 10, 0.30]
produto2 = [ "pera", 5, 0.75]
produto3 = [ "kiwi", 4, 0.98]
compras = [ produto1, produto2, produto3]
print(compras)
print(compras[0])
print(compras[0][0])
```

```
[['maçã', 10, 0.3], ['pera', 5, 0.75], ['kiwi', 4, 0.98]]
['maçã', 10, 0.3]
maçã
```

Interação com Listas Multidirecionais

```
produto1 = [ "maçã", 10, 0.30]
produto2 = [ "pera", 5, 0.75]
produto3 = [ "kiwi", 4, 0.98]
compras = [ produto1, produto2, produto3]
print("---------------------------------")
for Linha in compras:
    print("Produto: %s" % Linha[0])
    print("Quantidade: %d" % Linha[1])
    print("Preço: %5.2f" % Linha[2])
    print("---------------------------------")
```

Interação com Listas Multidirecionais

```
-----------------------------------
Produto: maçã
Quantidade: 10
Preço:  0.30
-----------------------------------
Produto: pera
Quantidade: 5
Preço:  0.75
-----------------------------------
Produto: kiwi
Quantidade: 4
Preço:  0.98
-----------------------------------
```

Exercícios:

3 - Altere o software do exercício anterior de forma que ele receba os dados do usuário para criar na lista de compras. O sistema deve oferecer ao usuário a possibilidade de exibir a lista de compras, adicionar e remover itens, além de mostrar os totais (valor total da compra e quantidade total de ítens).

Bibliografia

- MENEZES, Nilo Ney Coutinho - Introdução à Programação com Python: Algoritmos e Lógica de Programação Para Iniciantes, 3ª Edição – 2019, Editora: Novatec Editora, ISBN-10: 8575227181
- SHAW, Zed A – Aprenda Python 3 do jeito certo, 1ª Edição – 2019, Editora: Alta Books, ISBN: 978-85-508-0473-6.
- https://docs.python.org/pt-br/3/
- https://www.ime.usp.br/~leo/mac2166/2017-1/introducao_estrutura_basica_c_python.html
- http://python42.com.br/?p=176
- https://www.youtube.com/@CursoemVideo
- https://panda.ime.usp.br/cc110/static/cc110/