Curso de Computação 1
Introdução à Programação em Python

AULA 8

Estrutura de repetição: for

Na aula anterior estudamos como implementar a repetição de tarefas. O comando **while** é a forma genérica de se repetir execuções de um trecho de código. Porém, não é a única. Nesta aula, veremos uma segunda forma de se repetir a execução de um determinado trecho de código.

1. Apresentação do comando for

Na aula anterior, fomos apresentados ao **while** e vimos como implementar a repetição de tarefas com este comando. Vimos que uso do **while** requer uma certa atenção por parte do programador na definição do critério de parada e também na atualização das variáveis que expressam a condição de saída do **while**. Isto porque há sempre o risco de se implementar um *loop* infinito se o programador errar a expressão booleana que define a condição de saída ou a atualização das variáveis que expressam esta condição. Tudo que não queremos em programação é ter preocupações com pequenos detalhes que podem trazer grande impacto no nosso programa. Mais que isso, repetição de tarefas é algo bastante frequente na programação, o que significa dizer que quase sempre vamos precisar implementar estruturas de repetição em nossos programas. Por isso, foi disponibilizada uma segunda forma de se implementar a repetição de tarefas no Python: o comando **for**. Veja o vídeo abaixo e descubra como e quando podemos usar o comando **for** para se implementar a repetição da execução de um trecho de código.

Apresentação do comando *for*

Agora que vimos como usar o **for**, vejamos um trecho de código com o comando **for** funcionando na prática. O vídeo abaixo mostra a execução passo-a-passo e detalhada de uma das funções apresentadas no vídeo anterior. Veja o vídeo prestando atenção na atualização da variável interna do **for** em cada iteração. Sugerimos que após ver o vídeo, visite o link do Python Tutor e faça você mesmo o teste desta função, ou com o mesmo valor de entrada do exemplo, ou algum outro valor que considere mais interessante para teste. Se surgir alguma dúvida, anote e pergunte ao seu professor na aula síncrona.

Exemplo de execução do comando for

Exercício: Analise os códigos das funções vistas até agora (distribuído junto com este roteiro) e compare com os códigos das mesmas funções escrito com o comando while (distribuído junto com o roteiro de estudos anterior).

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “Introdução ao for”. Fique atento ao prazo de entrega dessa atividade!

2. Exemplos de código

Vamos agora ver alguns usos típicos do comando for. O primeiro exemplo mostra como podemos percorrer dados de tipos indexáveis (strings, tuplas e listas). Neste exemplo, a sintaxe do **for** é comparada com a do **while**. Veja o vídeo a seguir e perceba as vantagens no uso do **for**.

Percorrendo dados iteráveis

Um tipo de iteração bastante comum em programação é executarmos uma determinada tarefa uma quantidade pré-estabelecida de vezes, não estando esse número pré estabelecido necessariamente relacionado com os elementos de um dado iterável. Neste caso, simplesmente sabemos quantas vezes uma determinada sequência de ações precisa ser repetida. A implementação deste tipo de iteração com o **for** pode ser feito em conjunto com a função **range**, de forma a se simular um contador de iterações.

A função range tem como objetivo produzir um dado iterável a partir de 1, 2 ou 3 argumentos que recebe. Estes três argumentos têm papel similar aos três argumentos que podem ser usados com a operação de fatiamento (*início, fim,* e *incremento*). Enquanto na operação de fatiamento estes argumentos se referem aos índices usados para definir a fatia, na função range estes argumentos são os números inteiros que determinarão os elementos do iterável que será gerado: v*alor_inicial, valor_de_parada* e *incremento*.

As opções de uso da função são:

- range (valor_de parada): gera a sequência de valores que vai de 0 (zero) até valor_de_parada (exclusive).

  Exemplo:

  ```
  >>> range(8)
  ```

  Produz a sequência de valores: *0,1,2,3,4,5,6,7*

- range (valor_inicial, valor_de_parada) : gera a sequência de valores que vai de valor_inicial até valor_de_parada (exclusive).

  Exemplos:

  ```
  >>> range(5,12)
  ```

  Produz a sequência de valores: *5,6,7,8,9,10,11*

  ```
  >>> range(0,8)
  ```

  Produz a sequência de valores: *0,1,2,3,4,5,6,7*
  idêntica à sequência produzida por range(8)!

  ```
  >>> range(-10,-2)
  ```

  Produz a sequência de valores: *-10, -9, -8, -7, -6, -5, -4, -3*

- range (valor_inicial, valor_de_parada, incremento): gera a sequência de valores que vai de valor_inicial até valor_de_parada (exclusive), mas ao invés de ir de 1 em 1, vai de incremento em incremento, ou seja, soma-se incremento a cada valor para obter o próximo valor da sequência.

  Exemplos:

  ```
  >>> range(5,12,2)
  ```

  Produz a sequência de valores: *5,7,9,11*

  ```
  >>> range(0,8,3)
  ```

  Produz a sequência de valores: *0,3,6*

  ```
  >>> range(8,0,-1)
  ```

Produz a sequência de valores: *8, 7, 6, 5, 4, 3, 2, 1*

```
>>> range(0,8,-1)
```

Produz uma sequência de valores VAZIA!

```
>>> range(-2,-10,-1)
```

Produz a sequência de valores: -2, -3, -4, -5, -6, -7, -8, -9

É importante ressaltar que na versão 3.x do Python, a função range() retorna um objeto iterável e não um dado do tipo lista. Se desejado, podemos converter o retorno para listas com a função list().
Exemplo:

```
>>> list(range(8,0,-1) )
[8, 7, 6, 5, 4, 3, 2, 1]
```

Algumas referências interessantes sobre a função range:

- https://pythonacademy.com.br/blog/a-funcao-range-do-python
- https://pt.stackoverflow.com/questions/361275/para-que-serve-a-fun%C3%A7%C3%A3o-range-em-python

Veja o vídeo a seguir sobre como implementar no Python este tipo de iteração, usando o resultado da função range como o iterável sobre o qual o laço for vai atuar:

Uso do for para uma quantidade fixa de repetições

Por fim, o **for** se trata de uma segunda possibilidade de implementação de uma estrutura de repetição, assim como o **while**, que foi a primeira forma de implementarmos uma repetição em código. O vídeo a seguir discute as diferenças entre estes comandos e as situações onde o uso seria recomendado para ambas as opções, dado o tipo de loop que queremos implementar.

Diferenças entre o uso do **while** e do **for**

3. Possíveis erros e avisos

Curso de Computação 1
Introdução à Programação em Python

A grande vantagem do uso do **for** é a automatização de alguns padrões usados com estruturas de repetição, fugindo assim de alguns erros comuns quando se tenta implementar a repetição por meio do **while.** Entretanto, isso não significa que o uso do **for** esteja livre de erros. Mostraremos a seguir alguns exemplos de código que apresentarão problemas caso venham a ser executados. Preste atenção nestes exemplos para não repetí-los quando estiver programando.

- Exemplo 1: passagem de não-indexável para o **for**. O código a seguir mostra o código da função *somaParesComForV1* alterado. Nele, falta a função **range**, de forma que o elemento passado para o **for** é um número inteiro. Este código irá abortar com uma mensagem de erro quando tentar executar o loop pois o inteiro não é indexável, ou seja, não possui várias posições com valores diferentes para serem vistas uma a uma separadamente.

```
def somaParesV1(indexavel):
    '''soma os numeros pares de um indexavel
    tupla|lista -> int'''
    soma=0
    for i in len(indexavel):
        if indexavel[i]%2==0:
            soma=soma+indexavel[i]
    return soma
```

Corrigindo este erro, ficamos com o código a seguir:

```
def somaParesV2(indexavel):
    '''soma os numeros pares de um indexavel
    tupla|lista -> int'''
    soma=0
    for i in range(len(indexavel)):
        if indexavel[i]%2==0:
            soma=soma+indexavel[i]
    return soma
```

Mas esta versão ainda não é a ideal, pois não aproveita ao máximo as facilidades fornecidas pelo comando for no Python. Como neste problema apenas estamos interessados no valor

presente em cada posição e não usamos a informação sobre o índice onde o valor se encontrava, o código pode ficar ainda mais enxuto:

```
def somaParesV3(indexavel):
    '''soma os numeros pares de um indexavel
    tupla|lista -> int'''
    soma=0
    for elemento in indexavel:
        if elemento%2==0:
            soma=soma+elemento
    return soma
```

- Exemplo 2: retorno precoce com **for**. No exemplo abaixo, ainda usando uma versão alterada da função *somaParesComForV1,* o **return** está alinhado para ser executado dentro do **if** que, por sua vez, está dentro do **for**. Isto significa dizer que quando o **if** testar sua condição como verdadeira, o **return** será executado, o que fará com que o **for** seja interrompido, pois a função a que ele faz parte termina sua execução. O mesmo problema aconteceria se tivéssemos o **return** alinhado para ser executado dentro do **for**, neste caso fazendo com que o **for** execute apenas uma iteração. Note que no código apresentado na videoaula o **return** está alinhado para ser executado após o término do **for**. Tenha cuidado com este caso porque é um dos mais comuns em programadores inexperientes. Os editores de texto de python, como é o caso do editor de texto do IDLE, muitas vezes querendo ajudar, tentam adivinhar qual seria o alinhamento correto da linha seguinte e nem sempre acertam.

```
def somaParesV4(indexavel):
    '''soma os numeros pares de um indexavel
    tupla|lista -> int'''
    soma=0
    for elemento in indexavel:
        if elemento%2==0:
            soma=soma+elemento
            return soma
```

Atividade: Faça agora a atividade “Mais exemplos com for”, para exercitar seus conhecimentos. Fique atento ao prazo de entrega dessa atividade!

Curso de Computação 1
Introdução à Programação em Python

Prática em Programação

Após concluir as etapas anteriores deste roteiro, faça as atividades práticas desta aula, disponíveis no Google Classroom da turma.

Até a próxima aula!