Lição 4

Estruturas de repetição – comando **for**

1. Comando for com a função geradora **range**

Nas lições anteriores, lidamos com programas com estruturas sequenciais e condicionais. É comum o programa precisar repetir algum bloco várias vezes. É aí que as estruturas de repetição são úteis. Existem os comandos de repetição, ou laços, **for** e **while** no Python, que abordamos nesta lição.

O comando for repete (itera) um ou mais comandos sobre elementos de qualquer sequência. Por exemplo, qualquer string em Python é uma sequência de seus caracteres, então podemos iterar sobre eles usando para:

```
1 for caractere in "Olá":
2     print(caractere)
3 # Saída:
4 # Ó
5 # l
6 # á
```

Outro caso de uso para um laço for é repetir com a variação de uma variável inteira (**variável de controle**) em ordem crescente ou decrescente. Essa sequência de inteiros pode ser criada usando a função geradora de valores **range(valor_mínimo, valor_máximo, passo)**:

```
1 for i in range(5, 8, 1):
2     print(i, i ** 2)
3 print("Comando seguinte ao for")
4 # Saída:
5 # 5 25
6 # 6 36
7 # 7 49
8 # Comando seguinte ao for
```

A função **range(valor_mínimo, valor_máximo, passo)** gera uma sequência com os números **valor_mínimo, valor_mínimo+passo, ..., valor_máximo-1**. O número **valor_máximo** não é incluído. Por exemplo, **range(1, 11, 1)** gera a sequência: **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9** e **10**. (**11** não é incluído). A **variável de controle** pode ser usada para o que for necessário dentro do comando **for**.

Há uma forma reduzidas de uso da função geradora **range()** - **range(valor_mínimo, valor_máximo)**, em que o **passo** é implicitamente **1**:

```
1 for i in range(-1, 4):
2     print(i)
3 # Saída:
4 # -1
5 # 0
6 # 1
8 # 2
9 # 3
```

Outra forma reduzidas de uso do gerador **range()** - **range(valor_máximo)** que é caso em que implicitamente o **valor_mínimo** é definido como zero e o **passo** como **1** :

```
1 for i in range(3):
2     print(i)
3 # 0
4 # 1
5 # 2
```

Desta forma, podemos repetir alguma ação várias vezes:

```
1 for i in range(2 ** 2):
2     print("Alô, mundo!")
3 # Saída:
4 # Alô, mundo!
5 # Alô, mundo!
6 # Alô, mundo!
7 # Alô, mundo!
```

Da mesma forma que no comando **if**, a **indentação** é o que define quais instruções são controladas pelo **for** e quais não são.

A função geradora **range()** pode definir uma sequência vazia, como **range(-5)** ou **range(7, 3)**. Neste caso, o bloco interno ao **for** não será executado:

```
1 for i in range(-5):
2     print("Alô, mundo!") # Esta instrução print não será executada
```

Temos aqui um exemplo mais complexo para somar os inteiros de 1 a n, inclusive.

```
1 resultado = 0
2 n = 5
3 for i in range(1, n+1):
4     resultado += i # forma reduzida para resultado = resultado + i
5 print(resultado)
6 # Saída:
7 # 15
```

Preste atenção que o valor máximo no **range()** é **n+1** para tornar **i** igual a **n** no último **passo**.

Para iterar em uma sequência decrescente, podemos usar a forma estendida de **range()** com três argumentos - **range(valor_inicial, valor_final, passo)**. Quando omitido, o **passo** é implicitamente igual a **1**. No entanto, pode ser qualquer valor diferente de zero, inclusive negativo. O laço sempre inclui o **valor_inicial** e exclui o **valor_final** durante as repetições:

```
1 for i in range(10, 0, -2):
2     print(i)
3 # Saída:
4 # 10
5 # 8
6 # 6
7 # 4
8 # 2
```

2. Configurando a função **print()**

Por padrão, a função **print()** imprime todos os seus argumentos separando-os por um espaço e coloca implicitamente um caractere de **nova linha** ('**\n**') após eles para haver uma mudança de linha ao final da impressão. Esse comportamento pode ser alterado usando os argumentos de palavras-chave **sep** (separador) e **end** (finalizador), como é mostrado no exemplo a seguir:.

```
 1 print(1, 2, 3)                     # Muda de linha
 2 print(4, 5, 6)                     # Muda de linha
 3 print(1, 2, 3, sep=", ", end=". ")  # Não muda de linha
 4 print(4, 5, 6, sep=", ", end=". ")  # Não muda de linha
 5 print()                            # Muda de linha
 6 print(1, 2, 3, sep="", end=" -- ")  # Não muda de linha
 7 print(4, 5, 6, sep=" * ", end=".")  # Não muda de linha
 8 # Saída
 9 # 1 2 3
10 # 4 5 6
11 # 1, 2, 3. 4, 5, 6.
12 # 123 -- 4 * 5 * 6.
```

3. Comandos **for** aninhados

Qualquer comando, em qualquer quantidade, pode ser inserido dentro de um comando **for**, inclusive um comando **if** ou outro comando **for**. Tudo depende do algoritmo a ser executado.

Neste exemplo, é usado o **for** para desenhar um triângulo de asteriscos, com um número determinado de linhas, com o formato mostrado a seguir:

```
  *
 **
***
...
```

```
num_lin = int(input("Informe o número de linhas: "))
for linha in range(1, num_lin+1, 1):
    for coluna in range(num_lin-linha, 0, -1):
        print(" ", end="")  # imprime um espaço sem mudar de linha
    for coluna in range(1, linha+1, 1):
        print("*", end="")  # imprime um asterisco sem mudar de linha
    print()  # muda de de linha
# Saída para 4 linhas:
#    *
#   **
#  ***
# ****
```

Observe que antes dos asteriscos de cada linha são mostrados espaços em branco para haver o deslocamento dos asteriscos para a direita no desenho do triângulo.