Malhas de repetição com contador

No capítulo de malhas de repetição vimos casos em que precisamos **contar** quantas vezes o *loop* se repete, e parar quando a contagem atinge um certo valor. Em outras ocasiões, apenas precisamos de algum tipo de sequência numérica. Nestes casos, era normal utilizar uma variável de contador, incrementá-la em cada passo e utilizar seu valor como condição de parada. O exemplo abaixo imprime todos os números pares entre 0 e 100:

```
contador = 0
while contador < 100:
    print(contador)
    contador = contador + 2
```

Existe um meio de automatizar todas as operações envolvidas: atribuir um valor inicial, atribuir um valor final e realizar o incremento.

1.1 *Loops* do tipo "para"

Dizemos que o exemplo acima é um *loop* do tipo "para": para contador de 0 até 100 com passo 2 faça: imprima contador. Em Python, podemos criar esse tipo de *loop* utilizando os comandos *for* e *range*. O exemplo abaixo imprime os números de 0 até 9 na tela:

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

O código acima é equivalente ao seguinte código utilizando `while`:

```
contador = 0
while contador < 10:
    print(contador)
    contador += 1
```

A palavra "contador" é apenas uma variável. Ela não precisa ser criada previamente: qualquer nome utilizado nesta construção será automaticamente inicializado pelo *for*.

O programa acima atribui o valor inicial 0 à variável. Em seguida, ele executa tudo que vier dentro do *loop*, e ao chegar ao final, ele retorna ao início, soma 1 na variável e testa se o seu

valor atingiu o número entre parênteses. Caso não tenha atingido, ele repete a execução. Dizemos que aquele número é o valor final *exclusivo* (pois o *loop* exclui esse valor).

De forma geral, tudo que vier dentro de um `for contador in range(x)` irá executar "x" vezes. É o jeito fácil de dizer "repita essas linhas x vezes" em Python.

1.2 Parâmetros do *range*

Foi dito que *loops* do tipo "para" seguem a forma "para contador de X até Y passo Z faça:". No exemplo acima, os valores iniciais (0) e passo (1) foram atribuídos de forma automática. Caso eles sejam omitidos, 0 e 1 são os valores padrão, respectivamente. Porém, podemos determiná-los, se necessário. O exemplo abaixo inicia a impressão dos números em 1 ao invés de 0:

```
for contador in range(1, 10):
    print(contador)
```

Dizemos que esse *loop* possui valor inicial 1, valor final (exclusivo) 10 e passo 1.

O código acima é equivalente ao seguinte código utilizando `while`:

```
contador = 1
while contador < 10:
    print(contador)
    contador += 1
```

Assim como manipulamos o valor inicial e o final, podemos manipular também o passo. Veja o exemplo abaixo:

```
for contador in range(0, 100, 2):
    print(contador)
```

O código acima é equivalente ao seguinte código utilizando `while`:

```
contador = 0
while contador < 100:
    print(contador)
    contador += 2
```

Note que o resultado dele na tela é exatamente o mesmo do exemplo com *while* do início deste capítulo! Valor inicial 0, valor final (exclusivo) 100 e passo 2. Porém, não precisamos nos preocupar em criar o contador, atribuir valor inicial, incrementar e criar uma condição de parada. Apenas colocamos os números dentro do *range* e ele fez a mágica por nós.

Antes de finalizar, vamos reforçar: o comportamento de cada parâmetro passado para o *range* depende de quantos parâmetros foram passados e da ordem que eles foram passados:

1 parâmetro = valor final exclusivo

2 parâmetros = valor inicial, valor final exclusivo

3 parâmetros = valor inicial, valor final exclusivo, passo

Quantos exercícios de *while* você fez que podem ser resolvidos de maneira mais fácil com o *for*?

Dica: é possível utilizar o *for* para gerar sequências numéricas decrescentes também. Basta adotar valor final menor do que o inicial e incremento negativo.

```
for contador in range(20, 0, -1):
  print(contador)
```

Qual valor será excluído da sequência: o 20 ou o 0? Tente deduzir e execute o programa para ver se acertou!

1.3. Comandos de desvio de fluxo

Os comandos de desvio de fluxo que estudamos junto do *while* (*break*, *continue* e *else*) também funcionam da mesma maneira com o *for*. Algumas observações sobre eles:

break: irá encerrar o loop antes de atingir o fim da sequência

else: será executado caso um break seja executado e ignorado caso o loop chegue ao final da sequência

continue: encerra o passo atual e passa para o próximo **avançando na sequência** automaticamente