Aula 09

Estruturas de controle de repetição quantificadas

Prof. Jalerson Lima

Apresentação

Nessa aula vamos dar início aos estudos das estruturas de controle de repetição, que podem ser classificadas como estruturas de repetição quantificadas e condicionais. Nessa aula iremos abordar apenas o primeiro grupo, que são as estruturas de repetição quantificadas.

Objetivos

1. Compreender o conceito de estruturas de repetição quantificadas;
2. Aprender a utilizar as estruturas de repetição quantificadas disponíveis em Ruby;
3. Aprender a utilizar comandos para controlar a execução dos laços;
4. Praticar os conhecimentos adquiridos através de exercícios práticos.

1. Introdução

Para ilustrar a utilidade das estruturas de controle de repetição quantificadas, também conhecidas como laços, iremos usar o exercício proposto na Atividade 9.1. Tente resolver esse exercício com os conhecimentos que você já possui.

Atividade 9.1

Crie um *script* em Ruby que lê um número inteiro e mostra a tabuada de multiplicação desse número de 1 a 10.

Para resolver o problema proposto na Atividade 9.1, provavelmente você produziu um *script* parecido com o ilustrado no Exemplo de código 1.

```
 1  puts "Digite um número: "
 2  numero = gets.chomp.to_i
 3
 4  puts "#{numero} x 1 = #{numero * 1}"
 5  puts "#{numero} x 2 = #{numero * 2}"
 6  puts "#{numero} x 3 = #{numero * 3}"
 7  puts "#{numero} x 4 = #{numero * 4}"
 8  puts "#{numero} x 5 = #{numero * 5}"
 9  puts "#{numero} x 6 = #{numero * 6}"
10  puts "#{numero} x 7 = #{numero * 7}"
11  puts "#{numero} x 8 = #{numero * 8}"
12  puts "#{numero} x 9 = #{numero * 9}"
13  puts "#{numero} x 10 = #{numero * 10}"
14
15  gets
```

Exemplo de código 1 – Solução da Atividade 9.1

Observe que, para resolver esse problema usando apenas os conhecimentos que temos atualmente, precisamos escrever repetidas vezes uma ação específica: multiplicar o número informado pelo usuário por um outro número (linhas 4 a 13). Para esse problema específico, precisamos realizar essa ação apenas 10 vezes, mas e se o problema exigisse que executássemos a ação 100 vezes? Ou 1000 vezes? Iríamos escrever a mesma linha de código várias vezes? E pior: o problema pode exigir que a ação seja executada um número variável de vezes.

Para resolver problemas como esse, em que ações precisam ser feitas repetidas vezes, é que devemos utilizar as estruturas de controle de repetição quantificadas. Essas estruturas são ditas quantificadas porque elas executam um número determinado de vezes.

2. Estruturas de controle de repetição quantificadas em Ruby

A linguagem Ruby possui três estruturas de controle de repetição quantificadas: o `for`, o `each` e o `times`.

2.1 Estrutura de repetição: for

O `for` é uma estrutura de repetição muito comum e presente em quase todas as linguagens de programação. O Exemplo de código 2 ilustra a sintaxe do `for`.

```
1  for <variável> in <expressão>
2    // Código a ser executado repetidamente
3  end
```

Exemplo de código 2 – Sintaxe do `for`

O Exemplo de código 3 ilustra um exemplo de uso do `for`.

```
1  for i in 1..10
2    puts "O valor de i é #{i}"
3  end
4
5  gets
```

Exemplo de código 3 – Exemplo de uso do `for`

Conforme explicamos anteriormente, o `for` é uma estrutura de repetição que executa um trecho de código várias vezes. Cada execução do `for` é chamada de iteração. Portanto dizemos que um `for` que executa 10 vezes possui 10 iterações. Observe, no Exemplo de código 3, que declaramos a variável `i`. Essa variável só existe dentro do `for` e ela assume um valor entre 1 e 10 a cada iteração. Na primeira execução do `for`, o valor de `i` é 1; na segunda execução do `for`, o valor de `i` é 2; na terceira execução do `for`, o valor de `i` é 3, e assim sucessivamente. O último valor que a variável `i` vai receber é 10.

Ao executar o código apresentado no Exemplo de código 3, você deverá observar o resultado abaixo.

```
O valor de i é 1
O valor de i é 2
O valor de i é 3
O valor de i é 4
O valor de i é 5
O valor de i é 6
O valor de i é 7
O valor de i é 8
O valor de i é 9
O valor de i é 10
```

Observe que o trecho de código dentro do `for` foi executado 10 vezes, e a cada iteração foi apresentado o valor da variável `i`.

Atividade 9.2

Resolva o problema proposto na Atividade 9.1 utilizando o `for`.

O Exemplo de código 4 ilustra uma solução para o problema proposto na Atividade 9.2.

```
1  puts "Digite um número: "
2  numero = gets.chomp.to_i
3
4  for i in 1..10
5    puts "#{numero} x #{i} = #{numero * i}"
6  end
7
8  gets
```

Exemplo de código 4 – Solução do problema proposto na Atividade 9.2

Observe, no Exemplo de código 4, que a linha de código que se repetia no Exemplo de código 1 foi usada apenas uma vez dentro do `for`. O `for` vai se encarregar de executar o que está dentro dele 10 vezes, conforme o intervalo definido na linha 4 (`1..10`).

Atividade 9.3

Crie um *script* em Ruby que leia um número inteiro X e mostre os números pares entre 1 e X.

2.2 Estrutura de repetição: each

O `each` é uma estrutura de repetição muito parecida com o `for`, contudo existem duas formas de se escrever o `each`, apresentadas no Exemplo de código 5 e no Exemplo de código 6.

```
1  <expressão>.each do |<variável>|
2    // Trecho de código executado repetidamente
3  end
```

Exemplo de código 5 – Sintaxe da estrutura de repetição `each`

```
1  <expressão>.each { |<variável>|
2    // Trecho de código executado repetidamente
3  }
```

Exemplo de código 6 – Sintaxe da estrutura de repetição `each`

Não há diferença de comportamento entre as duas formas de construir o `each`, portanto escolha aquela que preferir. O Exemplo de código 7 ilustra um exemplo de uso do `each`: uma contagem de 1 a 10.

```
1  (1..10).each do |i|
2    puts "O valor de i é #{i}"
3  end
4
5  gets
```

Exemplo de código 7 – Exemplo de uso do `each`

Ao executar o Exemplo de código 7, você vai observar uma contagem de 1 a 10, pois esse foi o intervalo definido na linha 1 (`1..10`). O Exemplo de código 8 ilustra uma solução da Atividade 9.1 usando o `each`.

```
1  puts "Digite um número: "
2  numero = gets.chomp.to_i
3
4  (1..10).each do |i|
5    puts "#{numero} x #{i} = #{numero * i}"
6  end
7
8  gets
```

Exemplo de código 8 – Solução da Atividade 9.1 usando o `each`

Observe que, novamente, não há repetição de linhas de código, pois o `each` vai se encarregar de executar a linha 5 repetidamente (10 vezes).

Atividade 9.4

Crie um *script* em Ruby que leia um número inteiro X e mostre todos os números entre 1 e X que são divisíveis por 3 ou por 5.

2.3 Estrutura de repetição: times

Na realidade o `times` não é uma estrutura de controle de repetição, mas sim um método da classe *Integer*, conforme vamos estudar mais à frente quando estivermos aprendendo sobre Programação Orientada a Objetos. Contudo, o método `times` também é muito utilizado para executar trechos de código repetidamente. Assim como o `each`, também existem duas formas

de se construir o `times`. Observe a sintaxe do `times` no Exemplo de código 9 e no Exemplo de código 10.

```
1  <expressão>.times do
2    // Trecho de código executado repetidamente
3  end
```

Exemplo de código 9 – Sintaxe do `times`

```
1  <expressão>.times { |<variável>|
2    // Trecho de código executado repetidamente
3  }
```

Exemplo de código 10 – Sintaxe do `times`

Uma diferença entre as duas formas de construir o `times` é que a primeira forma, ilustrada no Exemplo de código 9, dispensa o uso de uma variável auxiliar. Para ilustrar o exemplo da contagem usando o `times`, observe o Exemplo de código 11.

```
1  10.times { |i|
2    puts "O valor de i é #{i}"
3  }
4
5  gets
```

Exemplo de código 11 – Exemplo de uso do `times`

Ao executar o Exemplo de código 11, você deverá observar o seguinte resultado.

```
O valor de i é 0
O valor de i é 1
O valor de i é 2
O valor de i é 3
O valor de i é 4
O valor de i é 5
O valor de i é 6
O valor de i é 7
O valor de i é 8
O valor de i é 9
```

Observe que, diferentemente das estruturas `for` e `each`, o `times` iniciou o valor da variável auxiliar em 0 (zero). Isso ocorre porque no `for` e no `each` nós podemos configurar o valor inicial e final da variável auxiliar, contudo não podemos fazer o mesmo no `times`. A variável auxiliar do `times` sempre irá começar com o valor 0 (zero).

O Exemplo de código 12 ilustra uma solução da Atividade 9.1 usando o `times`.

```
1  puts "Digite um número: "
2  numero = gets.chomp.to_i
3
4  11.times { |i|
5    puts "#{numero} x #{i} = #{numero * i}"
6  }
7
8  gets
```

Exemplo de código 12 – Solução da Atividade 9.1 usando o *times*

Observe, na linha 4, que precisamos usar `11.times`, para que o valor da variável auxiliar varie entre 0 e 10. Ao executar o Exemplo de código 12, você deverá observar algo parecido com o seguinte.

```
Digite um número:
2
2 x 0 = 0
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
```

2.4 Controlando a execução de laços

As execuções dos laços podem ser controladas através dos comandos `break`, `next` e `redo`. Cada uma tem uma utilidade específica para controle de execução das estruturas de repetição, conforme será apresentado a seguir.

2.4.1 O comando break

O comando `break` interrompe a execução do laço. O Exemplo de código 13 ilustra um exemplo de uso do comando `break`.

```
1  for i in 1..10
2    if i == 7
3      break
4    end
5    puts "O valor de i é #{i}"
6  end
7
8  gets
```

Exemplo de código 13 – Exemplo de uso do `break`

Observe, no Exemplo de código 13, que o `each` está configurado para executar 10 vezes. Dentro do `each` há um `if` para verificar se o valor da variável `i` é igual a 7. Se for, o `break` é executado, interrompendo a execução do `each`. O resultado da execução do Exemplo de código 13 é apresentado abaixo.

```
O valor de i é 1
O valor de i é 2
O valor de i é 3
O valor de i é 4
O valor de i é 5
O valor de i é 6
```

Observe que o laço é executado 6 vezes. Na 7ª execução do laço, no qual o valor de `i` é 7, a execução é interrompida com a execução do `break`. Vale salientar que, se houverem dois laços aninhados (um laço dentro do outro), o `break` irá interromper apenas a execução do laço interno. Observe o Exemplo de código 14, que ilustra esse cenário.

```
1  for i in 1..10
2    puts "i = #{i}"
3    for j in 1..10
4      puts "j = #{j}"
5      if j > 5
6        break
7      end
8    end
9  end
10
11 gets
```

Exemplo de código 14 – Exemplo de uso do `break`

O código apresentado no Exemplo de código 14 possui dois `for`, um externo (no qual a variável auxiliar é `i`) e um interno (no qual a variável auxiliar é `j`). Quando o `break` for executado na linha 6, ele irá interromper apenas a execução do `for` interno, mas não do externo.

2.4.2 O comando next

O comando `next` é usado para pular a execução de uma iteração. O Exemplo de código 15 ilustra um exemplo de uso do comando `next`.

```
1  for i in 1..10
2    if i == 7
3      next
4    end
5    puts "O valor de i é #{i}"
6  end
7
8  gets
```

Exemplo de código 15 – Exemplo de uso do `next`

Observe, Exemplo de código 15, que o `for` está configurado para executar 10 vezes. Na 7ª iteração, na qual o valor de `i` é 7, o comando `next` é executado, interrompendo a execução da 7ª iteração e pulando para a próxima. Abaixo é apresentado o resultado da execução do Exemplo de código 15. Observe a ausência do "O valor de i é 7".

```
O valor de i é 1
O valor de i é 2
O valor de i é 3
O valor de i é 4
O valor de i é 5
O valor de i é 6
O valor de i é 8
O valor de i é 9
O valor de i é 10
```

Vale salientar que, quando o comando `next` é executado dentro de um laço aninhado (um laço dentro de outro), ele irá pular a iteração apenas do laço interno, assim como ocorre com o `break`.

2.4.3 O comando redo

O comando `redo` reexecuta a iteração atual. Observe o comportamento do `redo` no Exemplo de código 16.

```
1  for i in 1..10
2    puts "O valor de i é #{i}"
3    if i == 3
4      redo
5    end
6  end
```

Exemplo de código 16 – Comportamento do `redo`

O `for` está configurado para executar 10 vezes (linha 1). Contudo, quando o valor de `i` for igual a 3, o `redo` será executado (linha 4). Isso fará com que a 3ª iteração seja executada infinitamente. Isso é chamado de *loop* infinito: um laço que nunca para de executar. Portanto, ao executar o Exemplo de código 16, não espere a execução terminar, apenas feche a janela. Abaixo é apresentado o resultado da execução do Exemplo de código 16.

```
O valor de i é 1
O valor de i é 2
O valor de i é 3
O valor de i é 3
O valor de i é 3
O valor de i é 3
O valor de i é 3
O valor de i é 3
...
```

Observe que as duas iterações são executadas normalmente, e ao chegar na 3ª iteração, o laço `for` fica preso num *loop* infinito.

Atividade 9.5

a) Faça um *script* em Ruby que leia um número *n* e mostre a tabuada de multiplicação de 1 a 10 deste número.
b) Faça um *script* em Ruby que mostre os números pares entre 1 e 100.
c) Faça um *script* em Ruby que mostre o somatório dos números pares entre 1 a N, onde N é um valor definido pelo usuário.
d) Desenvolva um *script* em Ruby que mostre todos os números entre 1 e 200 que são divisíveis por 3 ou por 5.
e) Desenvolva um *script* em Ruby que leia *n* números (o número *n* deve informado pelo usuário), e diga quantos são pares e quantos são ímpares. Imprima também a soma dos números pares, e a soma dos números ímpares.
f) Desenvolva um *script* em Ruby que, dado 2 números inteiros X e Y, calcule o valor de X^Y. Faça isso sem usar o operador de potenciação (**).
g) Faça um *script* em Ruby que calcule o fatorial de um número inteiro dado pelo usuário. O fatorial de um número é calculado através da multiplicação do próprio número pelos seus antecessores. Exemplo: o fatorial de 4 é 4 x 3 x 2 x 1 = 24.
h) Um número é primo se os únicos divisores dele são 1 e o próprio número. Faça um *script* em Ruby para ler um número inteiro positivo e determinar se ele é ou não um número primo.
i) No dia da estreia do filme "O Senhor dos Anéis", uma grande emissora de TV realizou uma pesquisa logo após o encerramento do filme. Cada espectador respondeu a um questionário no qual constava sua idade e a sua opinião em relação ao filme: 3 – excelente; 2 – bom; 1 – regular. Criar um *script* em Ruby que receba a idade e a opinião de 20 espectadores, calcule e imprima:
    a. A média das idades das pessoas que responderam excelente;
    b. A quantidade de pessoas que responderam regular;
    c. A percentagem de pessoas que responderam bom entre todos os espectadores analisados.
j) Crie um *script* em Ruby que imprime todas as tabuadas de multiplicação de 1 a 10.
k) Criar um *script* em Ruby que entre com 2 notas (de 0 a 100) de cada aluno de uma turma de 5 alunos, e imprima:
    a. A média de cada aluno;
    b. A média da turma;
    c. O percentual de alunos com média maior ou igual a 60.

Resumindo

Essa aula apresentou as estruturas de repetição quantificadas disponíveis em Ruby, bem como os comandos que permitem controlar a execução das iterações dessas estruturas. Além disso, foram apresentados exercícios para exercitar os conhecimentos transmitidos nessa aula.

Referências

POINT, T. Ruby Tutorial. **Tutorials Point**, 2015. Disponivel em: <http://www.tutorialspoint.com/ruby/>. Acesso em: 12 nov. 2015.

RANGEL, E. **Conhecendo Ruby**. [S.l.]: Leanpub, 2014.

SOUZA, L. **Ruby - Aprenda a programar na linguagem mais divertida**. 1ª. ed. São Paulo: Casa do Código, v. I, 2012.