Aula 10

Estruturas de controle de repetição condicionais

Prof. Jalerson Lima

Apresentação

Nessa aula iremos estudar as estruturas de repetição condicionais, ou simplesmente laços condicionais, que assim como os laços quantificados, são estruturas usadas para executar repetidamente trechos de código, contudo, os laços condicionais executam dependendo de uma condição.

Objetivos

1. Compreender a utilidade das estruturas de repetição condicionais;
2. Aprender quais e como funcionam os laços condicionais disponíveis em Ruby;
3. Exercitar os conhecimentos através de exercícios práticos.

1. Introdução

Na aula passada demos início ao estudo das estruturas de repetição, que são estruturas de controle de fluxo que permitem a execução de trechos de código de forma repetida. As estruturas de controle quantificadas executam um trecho de código um determinado número de vezes.

Nessa aula abordaremos as estruturas de controle de repetição condicionais, também conhecidas como laços condicionais, que também executam um trecho de código de forma repetida, contudo, a sua execução depende de uma condição.

Para ilustrar a utilidade das estruturas de repetição condicionais, observe a Atividade 10.1.

Atividade 10.1

Crie um *script* em Ruby que leia um número inteiro repetidas vezes. A repetição deve terminar quando o usuário digitar um número par.

Observe que, nesse caso, não é indicado utilizar uma estrutura de repetição quantificada, pois não é possível determinar, com antecedência, quantas vezes o *script* deverá solicitar o número para o usuário. Nesse caso a execução da estrutura depende de uma condição (o usuário digitar um número par), portanto a estrutura de repetição indicada para esse tipo de situação é a condicional.

Observe o Exemplo de código 1, que ilustra uma solução para a Atividade 10.1.

```
1  begin
2    puts "Digite um número par: "
3    numero = gets.chomp.to_i
4  end while numero % 2 != 0
5
6  puts "#{numero} é par!"
```

Exemplo de código 1 – Solução da Atividade 10.1

O `while` é uma estrutura de repetição condicional que executa enquanto a condição for verdadeira. Nesse caso a condição é `numero % 2 != 0`, que será falsa quando o usuário digitar um número par. O trecho de código a ser executado repetidamente se localiza entre o `begin` (linha 1) e o `end` (linha 4).

2. Estruturas de repetição condicionais em Ruby

A linguagem Ruby possui duas estruturas de repetição condicionais que funcionam de forma semelhante, e que podem ser escritas de três formas diferentes.

2.1 Estrutura de repetição: while

A estrutura de repetição *while* pode ser escrita de três formas distintas, a primeira delas está ilustrada no Exemplo de código 2.

```
1  while <condição>
2    // Trecho de código executado repetidamente
3  end
```

Exemplo de código 2 – Sintaxe do *while*

O trecho de código a ser executado repetidamente, localizado entre o `while` (linha 1) e o `end` (linha 3), será executado enquanto a `<condição>` for verdadeira. Tente resolver o exercício proposto na Atividade 10.2 utilizando o `while`.

Atividade 10.2

Crie um *script* em Ruby que leia um número inteiro repetidas vezes. A repetição deve terminar quando o usuário digitar um número múltiplo de 5.

Observe o Exemplo de código 3, que apresenta uma solução do exercício proposto na Atividade 10.2.

```
1  condicao = true
2  while condicao
3    puts "Digite um número múltiplo de 5: "
4    numero = gets.chomp.to_i
5    if numero % 5 == 0
6      condicao = false
7    end
8  end
9  puts "#{numero} é múltiplo de 5!"
```

Exemplo de código 3 – Solução da Atividade 10.2

No Exemplo de código 3, utilizamos a variável `condicao` para indicar se o `while` deve ser executado ou não. A variável `condicao` inicia com o valor `true` (linha 1), para que o `while` seja executado. Dentro do `while` (linhas 3 a 7), o *script* solicita que o usuário digite um número múltiplo de 5 (linha 3), e o valor é guardado na variável `numero` (linha 4). Em seguida, um `if` (linha 5) é usado para testar se o valor guardado na variável `numero` é, de fato, múltiplo de 5, e caso seja, o valor da variável `condicao` é alterado para `false` (linha 6) para que o `while` pare de executar.

A segunda forma de escrever o `while` é aquela apresentada no Exemplo de código 1. A sintaxe dessa segunda forma é apresentada no Exemplo de código 4.

```
1  begin
2    // Código a ser executado repetidamente
3  end while <condição>
```

Exemplo de código 4 – Sintaxe do *while*

A diferença entre essa sintaxe e aquela apresentada anteriormente (Exemplo de código 2) é que, utilizando dessa forma, o `while` executará pelo menos uma vez, independente a `<condição>` ser verdadeira ou falsa. Isso ocorre porque a avaliação da condição só ocorre no final (linha 3). Utilizando a sintaxe apresentada no Exemplo de código 2, caso a condição seja falsa no início da execução do `while`, ele não executará nenhuma vez.

Para melhor ilustrar a diferença entre as duas formas de escrita do *while*, observe e execute o Exemplo de código 5.

```
1  condicao = false
2
3  while condicao
4    puts "Executando o primeiro while..."
5  end
6
7  begin
8    puts "Executando o segundo while..."
9  end while condicao
```

Exemplo de código 5 – Diferenças entre as duas formas de escrever o *while*

Na linha 1, a variável `condicao` foi iniciada com `false`. Observe que o primeiro `while` (linhas 3 a 5) não executa nenhuma iteração pois `condicao` é `false`. O segundo `while` (linhas 7 a 9) executa uma vez mesmo com a variável `condicao` sendo `false`.

Por fim, o Exemplo de código 6 apresenta a sintaxe da terceira e última forma de se construir o *while*.

```
1  <código> while <condição>
```

Exemplo de código 6 – Sintaxe do *while*

Vale salientar que, usando a sintaxe apresentada no Exemplo de código 6, o trecho de código a ser executado repetidamente se limita a uma única linha de código.

2.2 Estrutura de repetição: until

A segunda estrutura de repetição condicional disponível em Ruby é o *until*. Assim como o *while*, o *until* também possui três formas distintas de se construir. Contudo, diferentemente do *while*, o *until* irá executar repetidamente enquanto a condição for falsa, e não verdadeira, como ocorre no *while*.

Observe a primeira forma de se escrever o *until* no Exemplo de código 7.

```
1  until <condição>
2    // Trecho de código a ser executado repetidamente
3  end
```

Exemplo de código 7 – Sintaxe do *until*

Para melhor ilustrar o *until*, observe o Exemplo de código 8, que apresenta uma solução para a Atividade 10.2 utilizando o *until* ao invés do *while*.

```
1  condicao = false
2  until condicao
3    puts "Digite um número múltiplo de 5: "
4    numero = gets.chomp.to_i
5    if numero % 5 == 0
6      condicao = true
7    end
8  end
9  puts "#{numero} é múltiplo de 5!"
```

Exemplo de código 8 – Solução da Atividade 10.2 com o *until*

Observe que o Exemplo de código 8 é bem similar ao Exemplo de código 3. Foram feitas apenas algumas mudanças no Exemplo de código 8 para que a solução utilizasse o *until* ao invés do *while*: na linha 1, o valor da variável `condicao` foi alterado para `false` para que o `until` seja executado; na linha 2, o `while` foi substituído pelo `until`; na linha 6, a variável `condicao` recebe `true` para que, quando isso ocorrer, o `until` pare de executar.

A segunda forma de se construir o *until* é ilustrada no Exemplo de código 9.

```
1  begin
2    // Código a ser executado repetidamente
3  end until <condição>
```

Exemplo de código 9 – Sintaxe do *until*

Observe que a segunda forma, apresentada no Exemplo de código 9, é bem semelhante à estrutura apresentada no Exemplo de código 4 com o *while*. A diferença entre elas é que o *while* executa enquanto a condição for verdadeira, e o *until* executa enquanto a condição for falsa.

Para melhor ilustrar essa segunda forma de construção do *until*, observe o Exemplo de código 10, que apresenta uma solução para a Atividade 10.1 utilizando o *until*.

```
1  begin
2    puts "Digite um número par: "
3    numero = gets.chomp.to_i
4  end until numero % 2 == 0
5
6  puts "#{numero} é par!"
```

Exemplo de código 10 – Solução da Atividade 10.1 com o *until*

Observe que, para resolver o problema proposto na Atividade 10.1 usando o *until*, bastaram duas mudanças na linha 4 do Exemplo de código 1: substituir o `while` pelo `until` e `!=` por `==` na condição, pois usando o *until*, o laço para de executar quando a condição for verdadeira.

A terceira e última forma de se escrever o `until` é a mais simples, pois só permite que uma única linha de código execute repetidamente. Confira a sintaxe dessa terceira forma no Exemplo de código 11.

```
1  <código> until <condição>
```

Exemplo de código 11 – Sintaxe do *until*

Novamente, a única diferença entre usar o `until` ou `while` é que, com o `while` o `<código>` será executado enquanto a `<condição>` for verdadeira, e com o `until` o `<código>` será executado enquanto a `<condição>` for falsa.

Vale salientar também que os comandos `break`, `next` e `redo`, apresentados na aula passada, também funcionam com os laços condicionais.

Atividade 10.3

Algumas questões dessa atividade não precisam ser resolvidas, necessariamente, utilizando estruturas de repetição condicionais.

a) Crie um *script* em Ruby que permita que o usuário digite vários números e imprima o triplo de cada número digitado. O *script* deve acabar quanto o número -999 for digitado;
b) Crie um *script* em Ruby que permita que o usuário digite vários números positivos. Quando um número negativo for digitado, o *script* deve apresentar quantos números positivos foram digitados;
c) Crie um *script* em Ruby que permita o usuário digitar vários números positivos. Quando um número negativo for digitado, o *script* deve imprimir a média dos números positivos digitados;
d) Crie um script em Ruby que permita o usuário digitar vários números. Quando o número 0 (zero) for digitado, o script deve informar quantos números entre 100 e 200 foram digitados pelo usuário;
e) Crie um script em Ruby que permita que o usuário digite os nomes de várias profissões. Quando o usuário teclar ENTER sem digitar nada, o script deve informar quantas vezes "dentista" foi digitado;
f) Crie um script em Ruby que permita que o usuário digite o sexo de várias pessoas ("m" para masculino ou "f" para feminino). Quando o usuário teclar ENTER sem digitar nada, o script deve informar quantas vezes foi digitado "m";
g) Crie um script em Ruby que permita o usuário digitar vários números. Após ler cada número, o script deve mostrar o quadrado desse número. O script deverá encerrar quando o usuário digitar um número múltiplo de 6;
h) Crie um script que permita o usuário digitar vários números. O script deve se encerrar quando o usuário digitar -999. Para cada número digitado, o script deve imprimir todos os seus divisores;
i) Dado um país A, com 5 milhões de habitantes e uma taxa de natalidade de 3% ao ano, e um país B com 7 milhões de habitantes e uma taxa de natalidade de 2% ao ano, crie um script em Ruby que calcule e imprima o tempo necessário para que a população do país A ultrapasse a população do país B;
j) Chico tem 1,50 metros e cresce 2 cm por ano, enquanto Juca tem 1,10 metros e cresce 3 cm por ano. Crie um script em Ruby que calcule e imprima quantos anos serão necessários para que Juca seja maior que Chico;
k) Uma empresa de fornecimento de energia elétrica faz a leitura mensal dos medidores de consumo. Para cada consumidor, são digitados os seguintes dados:
    - Número do consumidor;
    - Quantidade de kWh consumidos durante o mês;
    - Tipo (código) do consumidor: 1 – residencial (R$ 0,03 por kWh); 2 – comercial (R$ 0,05 por kWh); 3 – industrial (R$ 0,07 por kWh).

Os dados devem ser lidos até que seja encontrado um consumidor com o número 0 (zero). Crie um script em Ruby que calcule e imprima:

- O custo total para cada consumidor;
- O total de consumo para os três tipos de consumidor;
- A média de consumo dos tipos 1 e 2.

l) Criar um script em Ruby que leia vários números inteiros e apresente o fatorial de cada número. O script se encerra quando o usuário digitar um número menor do que 1. Dica: o fatorial de um número é a multiplicação dele com todos os seus antecessores. Exemplo: o fatorial de 5 é 5 x 4 x 3 x 2 x 1 = 120;

m) Crie um script em Ruby que permita que o usuário digite a idade de várias pessoas. O script deve parar quando uma idade negativa for digitada. O script deve mostrar:

- Total de pessoas com menos de 21 anos;
- Total de pessoas com mais de 50 anos;

n) Crie um script em Ruby que que leia vários números inteiros e imprima a quantidade de números primos dentre os números que foram digitados. O script acaba quando se digita um número menor ou igual a 0 (zero);

o) Crie um script que permita o usuário digitar vários números. Para cada número digitado, o script deve verificar se ele é um número triangular. Dica: um número é triangular quando é resultado do produto de três números consecutivos. Exemplo: 2 x 3 x 4 = 24. O script deve se encerrar quando o número 0 (zero) for digitado;

p) Crie um script em Ruby que permita o usuário digitar vários números. O script acaba quando se digita -9999. Por fim, o script deve apresentar o maior número digitado pelo usuário;

q) Crie um script em Ruby que permita o usuário digitar o número da conta e o saldo de várias pessoas. Após a digitação dos dados de cada conta, o script deve apresentar o número da conta e informar se o saldo é positivo ou negativo. O script deve terminar quando um número de conta negativo for digitado. Ao final, o script deve mostrar percentual de contas com o saldo negativo;

r) Crie um script em Ruby que leia vários números. A leitura se encerra quando o usuário digitar 0 (zero). O script deve mostrar os números que forem múltiplos de sua posição na sequência. Exemplo: supondo que o usuário digitou os números 3, 7, 8 e 16. O número 3 foi o 1º a ser digitado, o número 7 foi o 2º, o 8 foi o 3º e o 16 foi o 4º. Nesse exemplo, o script deve apresentar os valores 3 e 16, pois 1 (posição do número 3) é múltiplo de 3, e 4 (posição do número 16) é múltiplo de 16;

s) Crie um script em Ruby que calcule o Mínimo Múltiplo Comum (MMC) entre dois números lidos. Dica: o MMC de dois números é o menor número múltiplo dos dois números informados;

t) Crie um script em Ruby que calcule o Máximo Divisor Comum (MDC) entre dois números inteiros lidos;

u) Repare a seguinte característica do número 3025: 30 + 25 = 55 e 55^2 = 3025. Criar um script que possa ler vários números inteiros de 4 algarismos, um de cada vez, e diga se o número apresenta a mesma característica (repare que 3025 / 100 = 30 com resto 25). O script termina quando for lido um valor menor que 1000 ou maior que 9999;

v) Crie um script que leia vários números inteiros positivos e imprima a média dos números múltiplos de 3. O script deve se encerrar quando 0 (zero) for digitado;
w) Criar um script em Ruby que leia vários números inteiros positivos e imprima o produto dos números ímpares e a soma dos pares. O script deve terminar quando o número 0 (zero) for digitado;
x) Criar um script em Ruby que possa ler um conjunto de pedidos de compra e calcule o valor total da compra. Cada pedido é composto pelos seguintes campos:
    - Número do pedido;
    - Preço unitário;
    - Quantidade.

    O script deverá processar novos pedidos até que o usuário digite 0 (zero) como número do pedido;

y) Criar um script que leia a idade e sexo (0 para masculino e 1 para feminino) de várias pessoas. Calcule e imprima a idade média, total de pessoas do sexo feminino com idade entre 30-45 (inclusive), e o número total de pessoas do sexo masculino. O script termina quando se digita 0 (zero) para a idade;
z) Na Usina de Angra dos Reis, os técnicos analisam a perda de massa de um material radioativo. Sabendo-se que esse material perde 25% de sua massa a cada 30 segundos, criar um script em Ruby que leia um valor real que representa o peso do material, mostre o tempo necessário para que a massa desse material se torne menor que 10 gramas.

Resumindo

Essa aula apresentou as estruturas de repetição condicionais, que são usadas para execução de trechos de código de forma repetida e condicional. Iniciamos com as formas de construção do *while* e do *until*, apresentando as diferenças entre eles. Por fim, trabalhamos os conhecimentos através de exercícios práticos.

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.