Aula 03

Memória, variáveis e constantes

Prof. Jalerson Lima

Apresentação

Nessa aula aprenderemos alguns conceitos relativos às memórias do computador, e como utilizar variáveis e constantes para guardar informações na memória.

Objetivos

1. Entender alguns conceitos relativos à memória do computador;
2. Aprender como criar e utilizar variáveis e constantes.

1. Memória

Figura 1 - John von Neumann. Fonte: Wikipédia, 2015.

Os computadores atuais se baseiam numa arquitetura conhecida como Arquitetura de von Neumann (pronuncia-se Nôimánn), desenhada pelo matemático húngaro John von Neumann (Figura 1), que consiste nos seguintes componentes: memórias, unidade aritmética e lógica, unidade central de processamento, unidade de controle, além de interfaces de entrada e saída de dados (WIKIPÉDIA, 2015).

Nesta aula vamos nos deter a discutir sobre a memória do computador e como usá-la a fim de construir programas de computador mais sofisticados. É interessante notar como as memórias de computador foram desenvolvidas baseadas em dois tipos de memórias do cérebro humano: a memória de curto prazo e a memória de longo prazo.

Figura 2 - Memória RAM. Fonte: Portal No Mínimo (nominimo.com.br)

Assim como as memórias do cérebro humano, as memórias de computador são mecanismos que permitem ao computador guardar informações de forma temporária ou permanente. As memórias temporárias, também conhecidas como memória principal, são equivalentes às nossas memórias de curto prazo, que guardam informações temporariamente. Um exemplo desse tipo de memória é a RAM, ilustrada na Figura 2. Quando o fornecimento de energia da memória temporária é cortado, os dados guardados nela são perdidos. Mas qual seria a utilidade de uma memória que perde informações? A resposta para essa pergunta é simples: algumas informações não precisam ser guardadas de forma permanente. Por exemplo: quando você "decora" o endereço de um colega para ir visita-lo, você só precisa dessa informação até chegar à casa dele. Você não precisa dessa informação para o resto da sua vida.

Figura 3 - Hard Disk. Fonte: Wikipédia, 2015.

Já as memórias permanentes, também conhecidas como memória secundária ou de massa, são equivalentes às nossas memórias de longo prazo, e guardam informações permanentemente. As informações guardadas nesse tipo de memória só serão descartadas se você excluí-las. Um exemplo desse tipo de memória é o HD (Hard Disk), ilustrado na Figura 3. Quando o fornecimento de energia é cortado, os dados guardados nesse tipo de memória são preservados. Conforme mencionado anteriormente, esse tipo de memória equivale à nossa memória de longo prazo, que guarda informações por longos períodos de tempo.

É interessante notar que qualquer informação guardada da memória permanente precisa necessariamente passar pela memória temporária antes, ou seja, primeiro a informação é guardada na memória temporária, e só depois ela é copiada para a memória permanente. Esse processo também acontece no nosso cérebro: quando estamos aprendendo algo, as informações

são guardadas inicialmente na nossa memória temporária, portanto são mais passíveis de esquecimento. Na medida em que estudamos e nos esforçamos mais para aprender, as informações passam da memória temporária para a memória permanente, e a partir daí podemos dizer que de fato aprendemos.

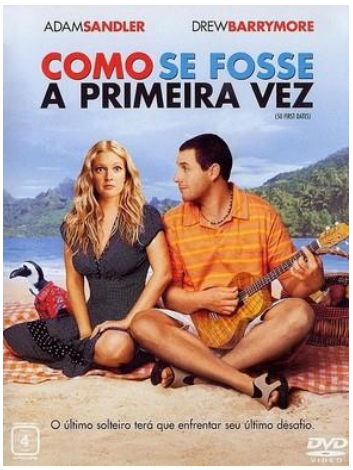

Figura 4 - Pôster do filme "Como se fosse a primeira vez". Fonte: filmow.com

Existe um filme chamado “Como se fosse a primeira vez” (Figura 4), estrelado por Adam Sandler e Drew Barrymore, que ilustra de forma bem-humorada a diferença entre a memória temporária e a memória permanente. No enredo do filme, Lucy (Drew Barrymore) sofreu um acidente que afetou seu cérebro, de forma que ela não consegue mais reter novas informações em sua memória permanente. Ela armazena novas informações apenas na sua memória temporária, que é apagada quando ela dorme. O bom humor do filme ocorre quando Henry (Adam Sandler) tenta conquista-la, mesmo sabendo que no dia seguinte ela não lembrará dele.

2. Variáveis

Agora vamos aprender como guardar informações na memória temporária do nosso computador, ou seja, a nossa memória RAM. As variáveis são o mecanismo usado para guardar informações na memória temporária, ou seja, nosso programa pode guardar dados dentro das variáveis, que ficarão armazenadas na memória temporária. Vale salientar que esses dados são perdidos quando o nosso programa encerrar ou o computador for desligado.

Em Ruby, criar uma variável e armazenar um dado dentro dela é muito simples. Abra o IRB, digite o código abaixo e aperte ENTER.

```
irb(main):001:0> numero = 3
=> 3
```

No exemplo anterior criamos uma variável chamada `numero` (sem acento) e armazenamos o valor 3 dentro dela.

O primeiro detalhe a ser percebido no exemplo anterior é que as variáveis possuem nomes. Os nomes são usados para diferenciar as variáveis, portanto lembre-se bem dos nomes das variáveis que você utilizou. Ao nomear uma variável, você precisa seguir algumas regras básicas.

- Não use acentos e caracteres especiais, prefira sempre nomes simples e que represente a informação que será guardada dentro dela;

- Nomes de variáveis não podem começar com número. Por exemplo, você não pode criar uma variável chamada `3x`;
- Se você precisar usar um nome composto (com mais de uma palavra) para a sua variável, use o *underline* (_) para separar as palavras. Por exemplo: `nome_completo`;
- As palavras abaixo são reservadas pela linguagem Ruby (DOC, 2015), portanto você não poderá usar nenhuma delas para nomear as suas variáveis.

__ENCODING__	case	false	redo	unless
__LINE__	class	for	rescue	until
__FILE__	def	if	retry	when
BEGIN	defined?	in	return	while
END	do	module	self	yield
Alias	else	next	super	
And	elsif	nil	then	
begin	end	not	true	
break	ensure	or	undef	

Ao longo da nossa disciplina, iremos estudar e conhecer a maioria das palavras reservadas. Abaixo você pode conferir alguns exemplos válidos de criação de variáveis.

```
irb(main):001:0> numero = 2
=> 3
irb(main):002:0> nome_completo = “Adam Sandler”
=> “Adam Sandler”
irb(main):003:0> idade = 26
=> 26
irb(main):004:0> preco_promocional = 2.33
=> 2.33
```

Perceba que estamos usando nomes de variáveis que indicam o que está guardado nelas. É muito comum ver alunos usando péssimos nomes para variáveis, conforme ilustramos abaixo.

```
irb(main):001:0> x = 2
=> 2
irb(main):002:0> z = “Adam Sandler”
=> “Adam Sandler”
irb(main):003:0> z2 = 26
=> 26
irb(main):004:0> x1 = 2.33
=> 2.33
```

Os nomes das variáveis ilustrados no exemplo anterior não indicam o que elas estão guardando, e, portanto, poderá confundi-lo ao construir seus programas.

Outro detalhe a ser percebido nos exemplos anteriores é que nós não apenas criamos novas variáveis, mas também guardamos valores dentro delas. Para guardar um valor dentro de uma variável, nós usamos o sinal de atribuição, que é o sinal de igual (=). Veja o exemplo abaixo.

```
irb(main):001:0> numero = 2
=> 2
```

No exemplo acima, criamos uma variável chamada `numero` que armazena o número inteiro 2.

Você lembra dos métodos `puts` e `gets.chomp` que aprendemos na aula anterior? Eles são comandos de entrada e saída de dados e também podem ser usados com variáveis! Observe o próximo exemplo.

```
irb(main):001:0> numero = 2
irb(main):002:0> puts numero
2
=> nil
```

No exemplo acima, criamos uma variável chamada `numero`, armazenamos o valor 2 dentro dela, e em seguida usamos o método `puts` para exibir o valor da variável na tela.

Como permitir que o usuário digite um dado no teclado e essa informação ser guardada numa variável? Observe o próximo exemplo.

```
irb(main):002:0> nome = gets.chomp
Drew Barrymore
=> “Drew Barrymore”
irb(main):003:0> puts nome
“Drew Barrymore”
=> nil
```

Na primeira linha do exemplo anterior usamos os métodos `gets.chomp` para permitir que o usuário digite o seu nome. Os métodos `gets.chomp` irão retornar (devolver como resultado) o valor digitado pelo usuário (uma *String*), e esse valor será guardado na variável `nome`. Na quarta linha do exemplo usamos o método `puts` para exibir o valor guardado na variável `nome`, que será o nome digitado por você.

Tudo bem até aqui? Se ficou com alguma dúvida, não deixe de nos procurar através do Ambiente Virtual de Aprendizagem, ok? Caso tenha entendido, observe o exemplo abaixo.

```
irb(main):001:0> idade = gets.chomp
29
=> 29
irb(main):002:0> idade_em_10_anos = idade + 10
TypeError: no implicit conversion of Fixnum into String
        from (irb):51:in `+'
        from (irb):51
        from C:/Ruby22-x64/bin/irb:11:in `<main>'
```

Algo deu errado. Você consegue perceber por que? Na aula anterior, vimos que o retorno, ou seja, o resultado dado pelos métodos `gets.chomp` é uma *String*. Portanto, apesar do usuário estar digitando um valor numérico (sua idade), essa informação será guardada na variável `idade` como uma *String*. Quando tentamos calcular a idade do usuário daqui a 10 anos, tentamos somar a variável idade (que guarda uma *String*) com o valor inteiro 10, e como já vimos na aula passada, não é possível somar uma *String* com um número. Portanto, para que o exemplo anterior funcione, é preciso converter o valor digitado pelo usuário para inteiro, e só depois disso poderemos soma-lo com 10. Observe abaixo o exemplo corrigido.

```
irb(main):001:0> idade = gets.chomp.to_i
29
=> 29
irb(main):002:0> idade_em_10_anos = idade + 10
=> 39
irb(main):003:0> puts idade_em_10_anos
39
=> nil
```

Lembra do método `to_i` que aprendemos na aula anterior? Ele está sendo usado na primeira linha do exemplo para converter o retorno (resultado) do método `chomp`, que é uma *String*, em um número inteiro. Portanto o valor guardado na variável idade é um inteiro. Agora nós podemos fazer a operação de soma entre a variável idade e o valor 10.

Em alguns momentos essa questão dos tipos das variáveis, ou seja, os tipos de dados que as variáveis guardam, pode ficar confusa. Isso ocorre porque Ruby é uma linguagem de "tipagem" dinâmica. Lembra que mencionamos isso na nossa primeira aula? Vejamos a definição que o (WIKIPÉDIA, 2015) apresenta sobre tipagem dinâmica.

> "Tipagem dinâmica é uma característica de determinadas linguagens de programação, que não exigem declarações de tipos de dados, pois são capazes de escolher que tipo utilizar dinamicamente para cada variável, podendo alterá-lo durante a compilação ou a execução do programa. Algumas das linguagens mais conhecidas a utilizarem tipagem dinâmica são: Python, Perl, Ruby, PHP, FoxPro e Lisp. Contrasta com a tipagem estática, que exige a declaração de quais dados poderão ser associados a cada variável antes de sua utilização, por exemplo: C, C++, Java, entre outras." (WIKIPÉDIA, 2015).

Ruby é uma linguagem de tipagem dinâmica, portanto você não precisa definir qual tipo de dado será guardado na variável ao cria-la. Um outro exemplo de tipagem é a estática, que exige que o programador, ao criar uma variável, informe qual tipo de dado será guardado nela. Observe o exemplo de código abaixo escrito na linguagem Java. Não tente fazer isso no IRB, pois esse exemplo está escrito em outra linguagem.

```
int idade = 29;
```

Quais diferenças você consegue perceber entre a criação de uma variável em Java e em Ruby? A primeira delas é a presença do `int` antes do nome da variável. Isso indica que essa

variável irá guardar um número inteiro, portanto ela não será capaz de guardar outros tipos de dados. A segunda diferença é a presença do ponto-e-vírgula no final da linha. Em Java, o final de cada instrução deve ser marcado com ponto-e-vírgula.

Portanto em linguagens de tipagem dinâmica como Ruby, você não precisa definir o tipo da variável ao criar uma. Já em linguagens de tipagem estática como Java, você precisa definir o tipo da variável ao cria-la. Em Ruby, você pode inclusive guardar diferentes tipos na mesma variável, conforme ilustra o exemplo abaixo.

```
irb(main):001:0> variavel = 3
=> 3
irb(main):002:0> variavel = “Ruby usa tipagem dinâmica”
=> "Ruby usa tipagem din\x83mica"
```

Observe que usamos a mesma variável para guardar um número inteiro, e em seguida guardar uma *String*. Porém toda essa flexibilidade tem um preço: em alguns momentos pode ficar difícil saber qual tipo de dado está sendo guardado em cada variável. Para atenuar esse problema, existem meios de descobrir qual tipo de dado está guardado em cada variável. Observe o exemplo abaixo.

```
irb(main):001:0> variavel_desconhecida = 20
=> 20
irb(main):002:0> variavel_desconhecida.class
=> Fixnum
```

No exemplo de código acima, usamos o método `class` na variável para descobrir o tipo de dado guardado na variável `variavel_desconhecida`. Perceba que o retorno (resultado) do método foi `Fixnum`, que é um tipo de dado inteiro. Observe mais exemplos abaixo.

```
 irb(main):001:0> variavel_desconhecida = “Tipo de dado desconhecido”
=> “Tipo de dado desconhecido”
irb(main):002:0> variavel_desconhecida.class
=> String
```

Atividade 3.1

Usando o IRB, crie uma variável chamada `nome` e atribua o seu nome a ela. Crie outra variável chamada `idade` e atribua a sua idade a ela. Em seguida mostre os valores de `nome` e `idade` na tela.

Atividade 3.2

Usando os métodos apresentados nessa aula, descubra os tipos de dado armazenados nas variáveis `nome` e `idade` criadas na Atividade 3.1.

3. Constantes

Uma constante é como uma variável, ou seja, um espaço na memória do computador na qual é possível guardar informações. Nas variáveis esse valor pode mudar, enquanto nas constantes esse valor, tecnicamente, não pode ser alterado (por isso o nome 'constantes'). Contudo o interpretador Ruby permite que você altere valores de constantes, mas ele dará um aviso para lhe alertar que você está alterando o valor de uma constante (GURU-SP, 2015).

Para criar uma constante, basta criar uma 'variável' com a primeira letra maiúscula. Observe o exemplo abaixo.

```
irb(main):001:0> Minha_constante = 2
=> 2
irb(main):001:0> Minha_constante = 3
(irb):3: warning: already initialized constant Minha_constante
(irb):1: warning: previous definition of Minha_constante was here
```

No exemplo anterior criamos uma constante chamada `Minha_constante` na primeira linha e atribuímos o valor 2 a ela. Em seguida, na linha 3, alteramos o valor da constante para 3, e recebemos o *warning* (alerta) do Ruby dizendo que a constante já havia sido criada, e que o valor dela foi definido na linha 1.

Em Ruby e em várias outras linguagens de programação existem as convenções. Convenções são normas que são adotadas pelos desenvolvedores para melhor organizar o código. Mas as convenções não são regras da linguagem, portanto se você não as segui-las, seu código funcionará perfeitamente, contudo ele não irá atender a essa convenção (acordo). É uma boa prática de programação seguir todas as convenções da linguagem, e aqui vai uma delas: por convenção, as constantes são escritas com todas as letras em maiúsculas. Por exemplo: `MINHA_CONSTANTE`, `IDADE`, `NOME`, etc.

Atividade 3.3

Crie uma constante chamada `IDADE` e atribua a sua idade a ela. Em seguida, mostre na tela o valor da constante criada por você usando o método `puts`.

Resumindo

Nessa aula aprendemos alguns conceitos relativos à estrutura e ao funcionamento das memórias do computador. Também aprendemos como utilizar a memória temporária do computador através de variáveis, que são importantes mecanismos para armazenamento de dados.

Referências

DOC, R. Keywords. **Ruby Doc**, 2015. Disponivel em: <http://ruby-doc.org/core-2.2.3/doc/keywords_rdoc.html>. Acesso em: 24 out. 2015.

FOUNDATION, P. S. Lexical Analysis. **PyDoc**, 2015. Disponivel em: <https://docs.python.org/3/reference/lexical_analysis.html>. Acesso em: 13 out. 2015.

GURU-SP. Constantes. **Tutorial de Ruby do GURU-SP**, 2015. Disponivel em: <http://guru-sp.github.io/tutorial_ruby/constantes.html>. Acesso em: 27 out. 2015.

WIKIPÉDIA. Arquitetura de von Neumann. **Wikipédia**, 2015. Disponivel em: <https://pt.wikipedia.org/wiki/Arquitetura_de_von_Neumann>. Acesso em: 11 out. 2015.

WIKIPÉDIA. Sistema de tipos. **Wikipédia**, 2015. Disponivel em: <https://pt.wikipedia.org/wiki/Sistema_de_tipos>. Acesso em: 13 out. 2015.