Aula 02

IRB, tipos de dados e operações de entrada e saída

Prof. Jalerson Lima

Apresentação

Nessa aula vamos aprender um pouco sobre o IRB, e como ele pode ser útil para introduzir alguns conceitos sobre a linguagem, como os tipos de dados disponíveis em Ruby e como realizar operações de entrada e saída de dados.

Objetivos

1. Praticar os conceitos apresentados com o IRB;
2. Aprender sobre os tipos de dados disponíveis em Ruby;
3. Aprender como realizar operações de entrada e saída de dados.

1 IRB

Quando você fez a instalação do Ruby no seu sistema operacional, também foi instalado um programa chamado IRB. O IRB (*Interactive Ruby*) é uma ferramenta que nos permite executar código Ruby e receber a resposta desses comandos imediatamente, muito útil quando queremos executar comandos simples para aprender a linguagem. Ele se parece muito com um terminal de comandos, como o *prompt* de comandos do Windows (DOC, 2015).

Abra o IRB no seu Windows, que deve se parecer com o programa ilustrado na Figura 1.

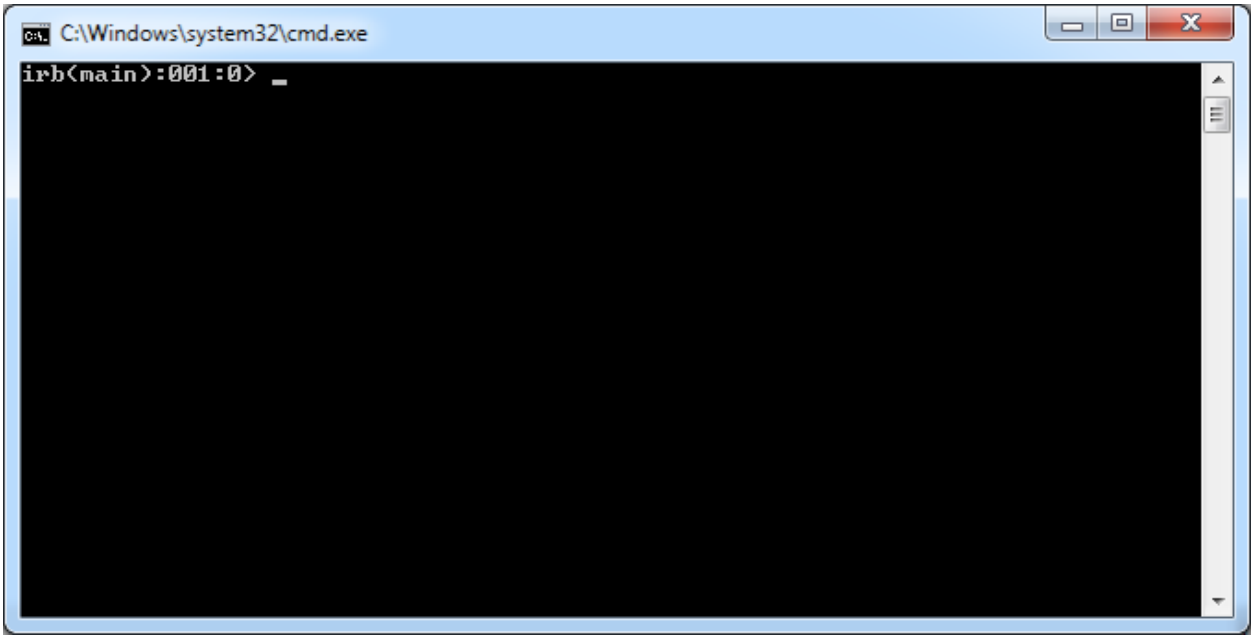

Figura 1 – IRB

Perceba que nós podemos escrever na primeira linha, na qual aparece `irb(main):001:0>`. Vamos explorar um pouco o IRB e a linguagem Ruby? Digite o valor 1 e aperte ENTER, conforme ilustrado abaixo.

```
irb(main):001:0> 1
=> 1
```

Observe a resposta do IRB: `=> 1`. Primeiramente, o sinal `=>` indica que o que aparece em seguida é o resultado de um comando, trecho de código ou expressão dada por você. Nesse caso, você digitou 1 e a resposta foi 1. O que acabou de acontecer? O IRB avalia expressões, trechos de código e instruções em Ruby, executa e dá alguma resposta. Nesse caso, a expressão que fornecemos foi 1 e a resposta, naturalmente, foi 1 porque não houve nenhuma operação matemática ou processamento a ser feito. Vamos tentar algo diferente, digite 1 + 1 e tecle ENTER.

```
irb(main):001:0> 1 + 1
=> 2
```

Nesse segundo exemplo fizemos uma operação matemática de soma entre dois números, e o resultado foi a soma deles. Vamos tentar algo um pouco mais complexo?

```
irb(main):001:0> 2 + 1 -
irb(main):002:0*
```

O que aconteceu? Observe que, na segunda linha, o caractere `>` foi substituído por `*`. Mas por que isso aconteceu? Porque nós digitamos uma expressão incompleta na primeira linha. Conforme vamos ver em uma aula futura, os operadores de soma e subtração precisam de dois operandos (um de cada lado do operador). Portanto está faltando um operando ao lado do sinal de subtração. O IRB ainda não executou a expressão que digitamos porque ela ainda está incompleta. A segunda linha nos permite digitar o que falta da expressão. Digite 3 na segunda linha e tecle ENTER, conforme o exemplo abaixo.

```
irb(main):001:0> 2 + 1 -
irb(main):002:0* 3
=> 0
```

Pronto! Agora o operador de subtração tem dois operandos (1 e 3). Pode não parecer, mas o que você está começando a fazer é construir seus próprios programas de computador. Parabéns! Vamos continuar nossos estudos aprendendo sobre os tipos de dados disponíveis na linguagem Ruby, dessa forma poderemos construir programas um pouco mais complexos.

Atividade 2.1

Use o IRB para realizar operações matemáticas usando os operadores `+` (soma), `-` (subtração), `*` (multiplicação) e `/` (divisão).

2 Tipos de dados

Quando estamos desenvolvendo um sistema de software, precisamos lidar com dados a todo momento. Conforme ilustrado na Figura 2, a linguagem Ruby suporta um conjunto de onze tipos de dados organizados de forma hierárquica.

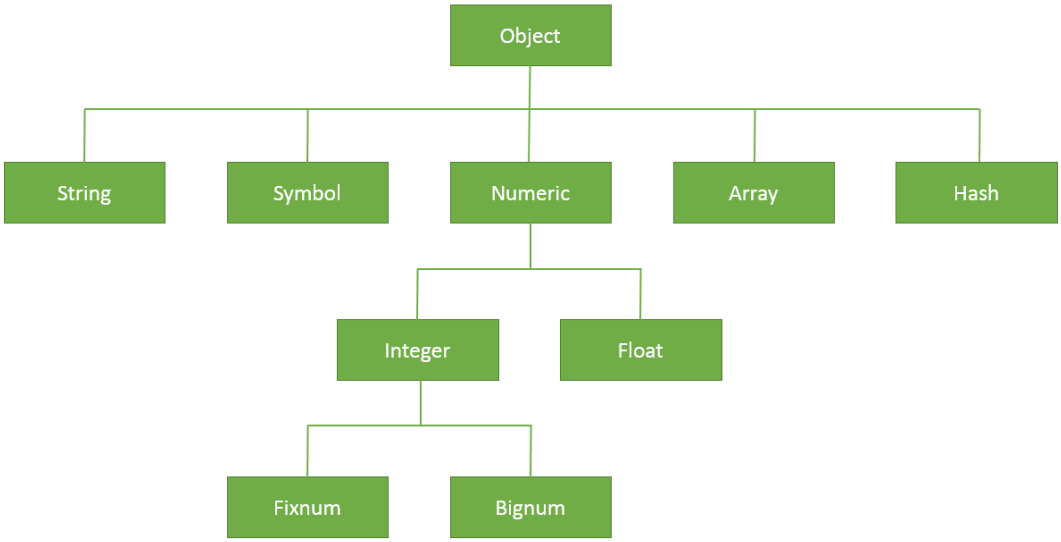

Figura 2 - Tipos de dados

O que podemos entender com a Figura 2? Todos os tipos de dados são subtipos de *Object*, existem dois subtipos numéricos (*Numeric*): *Integer* e *Float*; e dois subtipos inteiros (*Integer*): *Fixnum* e *Bignum*. Não se preocupe com todos os tipos agora, pois vamos aprendê-los aos poucos. Vamos continuar o estudo da nossa linguagem com os tipos de dados numéricos (*Integer* e *Float*) e texto (*String*). Os outros tipos de dados serão apresentados em aulas futuras.

2.1 Tipos de dados numéricos

Conforme mencionado anteriormente, existem dois tipos de dados numéricos: inteiro (*Integer*) e real (*Float*).

Números inteiros (*Integer*) são aqueles que não tem parte fracionária. Veja alguns exemplos de números inteiros abaixo. Quais informações poderiam ser representadas por números inteiros? A sua idade, o dia, o mês e o ano do seu nascimento, a quantidade de irmãos que você tem, etc.

```
irb(main):001:0> 1
=> 1
irb(main):002:0> 2
=> 2
irb(main):003:0> 10
=> 10
irb(main):004:0> 34
=> 34
```

Em Ruby, os números inteiros ainda podem ser classificados em *Fixnum* e *Bignum*. *Fixnum* é o tipo de dado para números inteiros pequenos (até 4.611.686.018.427.387.903), enquanto o *Bignum* é destinado para números inteiros grandes (a partir de 4.611.686.018.427.387.904) (SHAUGHNESSY, 2015).

Números reais (*Float*) são aqueles que tem parte fracionária, por exemplo: 3.14, 21.2, 4.583, etc. Observe que usamos o ponto ao invés da vírgula! Nas linguagens de programação em geral, e isso vale para Ruby também, nós devemos escrever 3.14 e não 3,14. Alguns exemplos de números reais são a quantidade de dinheiro que você tem em sua conta bancária, o seu peso, a distância que você percorre quando corre, etc.

```
irb(main):001:0> 3.14
=> 3.14
irb(main):002:0> 21.2
=> 21.2
irb(main):003:0> 4.583
=> 4.583
```

2.2 Texto (String)

Inicialmente vale salientar que quando queremos nos referir a texto, não usamos a palavra "texto", e sim "*String*". Portanto de agora em diante, quando nos referirmos a *String*, você já sabe que estamos nos referindo a texto. Esse tipo de dado tão comum deve ser escrito usando aspas simples ou duplas no início e no final do valor (SOUZA, 2012). Veja alguns exemplos abaixo.

```
irb(main):001:0> "Ser ou não ser, eis a questão!"
=> "Ser ou n\xC6o ser, eis a quest\xC6o"
irb(main):002:0> 'Hasta la vista, baby'
=> "Hasta la vista, baby"
```

Observe que no primeiro exemplo usamos aspas duplas para definir que `"Ser ou não ser, eis a questão!"` é um dado do tipo *String*. Veja também a reposta dada pelo IRB: `"Ser ou n\xC6o ser, eis a quest\xC6o"`. Percebeu que a resposta dada pelo IRB foi um pouco estranha? Não se preocupe com isso agora, isso se deve aos caracteres especiais que usamos na *String*.

No segundo exemplo usamos aspas simples para indicar que `'Hasta la vista, baby'` é uma *String*. Os dois exemplos têm o mesmo efeito e ambos são S*tring*. O cuidado que precisamos ter é, quando iniciarmos uma S*tring* com aspas duplas, ela precisa ser finalizada com aspas duplas. Quando iniciarmos com aspas simples, precisamos finalizar com aspas simples. Alguns exemplos de *String* que podem ser usadas num programa de computador são o seu nome, o seu endereço, um poema, etc.

2.3 Conversões de tipos

Tudo entendido até aqui? Então temos uma pergunta para você: "1" é um inteiro ou é uma *String*? E agora? O que você acha? "1" é uma *String*! Porque ele está entre aspas duplas. Se não estivesse entre aspas duplas, seria um número inteiro. Agora faça o seguinte exemplo no IRB.

```
irb(main):001:0> "1" + 1
TypeError: no implicit conversion of Fixnum into String
        from (irb):9:in `+'
        from (irb):9
        from C:/Ruby22-x64/bin/irb:11:in `<main>'
```

Recebemos a nossa primeira mensagem de erro. Consegue imaginar a causa do problema? O que ocorreu foi um erro de tipo (TypeError): nós usamos o operador de soma (+) com uma *String* ("1") e um número inteiro (1). Mas isso faz algum sentido? Faria sentido tentar somar "a" + 1? Claro que não! Apesar da *String* "1" conter um valor numérico escrito, para o Ruby "1" é uma *String*, e não é possível somar uma *String* com um número! Portanto, se tivermos um valor numérico escrito numa *String*, primeiro precisamos converter essa *String* para número.

Isso é o que chamamos de conversão de tipos. Ou seja, transformar o dado de um tipo e um dado de outro tipo. Por exemplo: transformar "1" (*String*) em 1 (*Integer*), 10 (*Integer*) em "10" (*String*), '3.14' (*String*) em 3.14 (*Float*), etc. Para converter tipos, devemos usar os métodos `to_i`, `to_f` e `to_s`.

Método	Descrição	Exemplo
<valor>.to_i	Converte <valor> para um número inteiro (Integer)	"10".to_i
<valor>.to_f	Converte <valor> para um número real (Float)	"3.14".to_f
<valor>.to_s	Converte <valor> para uma *String*	12.to_s

Observe os exemplos abaixo.

```
irb(main):001:0> "10".to_i
=> 10
irb(main):002:0> "3.14".to_f
=> 3.14
irb(main):003:0> 30.to_s
=> "30"
irb(main):004:0> 3.to_f
=> 3.0
```

Agora nós podemos corrigir o exemplo apresentado anteriormente.

```
irb(main):001:0> “1”.to_i + 1
=> 2
```

Entendeu? O método `to_i` converteu “1” (*String*) em 1 (*Integer*) e depois a operação de soma foi executada, resultando em 2. Observe o próximo exemplo.

```
irb(main):001:0> “a”.to_i
=> 0
```

No exemplo anterior, tentamos converter a *String* “a” para um valor inteiro. Obviamente isso não faz nenhum sentido, porque não é possível converter uma letra num número. Quando tentamos converter um valor não-numérico para *String*, o Ruby sempre nos dá zero como resultado.

3 Entrada e saída de dados

As operações de entrada de dados são aquelas que permitem ao usuário passar informações para o programa (entrada), e as operações de saída são aquelas que permitem o programa apresentar informações para o usuário. Vamos iniciar os estudos com a operação de saída de dados, para que possamos fazer o programa mais famoso do mundo: o Olá Mundo!

3.1 Saída de dados

Conforme explicado anteriormente, a operação de saída de dados permite que o programa apresente informações para o usuário. Para fazer isso, devemos usar o método `puts`. Mas o que é um método? Por enquanto pense no método como sendo um comando que irá fazer alguma tarefa para você. Nesse caso, o método `puts` apresenta alguma informação na tela do usuário. As informações apresentadas na tela são aquelas que você digitar ao lado do método.

Vamos fazer o programa “Olá Mundo!” para exemplificar o uso do método `puts`. Digite o seguinte comando no IRB e depois tecle ENTER.

```
irb(main):001:0> puts “Olá mundo!”
Olá mundo!
=> nil
```

Observe que usamos o método `puts`, e logo em seguida digitamos a *String* `“Olá mundo!”`. Quando fazemos isso, estamos informando para o método `puts` que desejamos que ela mostre `“Olá mundo!”` na tela, e foi exatamente isso o que aconteceu. Observe também que apareceu `=> nil`. `nil` é um valor especial que significa nulo, ou seja, algo inexistente. Ele foi exibido porque o método `puts` sempre retorna (dá como resultado) `nil`, além de mostrar a informação na tela.

Caso tenha ocorrido algum erro, verifique se você digitou corretamente, pois o código só será executado se ele estiver sintaticamente correto, ou seja, se ele estiver bem construído e formatado. Observe o que acontece quando digitamos algo errado.

```
irb(main):001:0> put “Olá mundo!”
NoMethodError: undefined method `put' for main:Object
        from (irb):27
        from C:/Ruby22-x64/bin/irb:11:in `<main>'
```

Novamente recebemos um erro. Você consegue perceber o que fizemos de errado? Nós escrevemos `put` ao invés de `puts`! Nesse caso, houve um *NoMethodError*, dizendo que o método `put` não foi encontrado.

3.2 Entrada de dados

A entrada de dados é uma operação que permite que o usuário forneça dados para o programa. Essa operação é realizada usando o método `gets`. Observe o exemplo abaixo.

```
irb(main):001:0> gets
Um texto qualquer
=> “Um texto qualquer\n”
```

No exemplo anterior, usamos o método `gets` para permitir que o usuário digite algo. Ao executar o método `gets`, o Ruby fica aguardando que o usuário digite algo na linha seguinte e tecle ENTER. No exemplo digitamos a *String* `“Um texto qualquer”` (sem aspas), e o resultado foi a *String* que digitamos seguido de `\n`. Mas de onde veio esse `\n`? Esse é um caractere especial que significa nova linha, e ele foi enviado pelo nosso teclado quando apertamos ENTER. E como evitamos isso? Usamos o método `chomp` sobre o retorno (resultado) do método `gets` da seguinte forma.

```
irb(main):001:0> gets.chomp
“Um texto qualquer”
=> “Um texto qualquer”
```

Pronto, dessa forma o método `gets` irá retornar (dará como resultado) aquilo que o usuário digitar no teclado. O método `chomp` será executado sobre o retorno do método `gets` (aquilo que o usuário digitou), e irá remover caracteres como o de nova linha. Vale salientar que os métodos `gets` e `chomp` sempre retornam (dão como resultado) um dado do tipo *String*. Não se preocupe neste momento em entender como capturar o valor digitado pelo usuário e realizar algum processamento com ele, pois isso será tema da nossa próxima aula.

Atividade 2.2

Use o IRB para escrever o primeiro programa de todo programador: o Olá Mundo! Abra o IRB no seu computador e escreva `puts "Olá mundo!"`.

Resumindo

Nessa aula aprendemos como utilizar o IRB para executar código Ruby. Também aprendemos sobre os tipos de dados disponíveis na linguagem, bem como o que eles representam e como usá-los. Por fim aprendemos como realizar operações de entrada e saída de dados em Ruby.

Referências

DOC, R. IRB. **Ruby Doc**, 2015. Disponivel em: <http://ruby-doc.org/stdlib-2.2.3/libdoc/irb/rdoc/IRB.html>. Acesso em: 24 out. 2015.

SHAUGHNESSY, P. How Big is a Bignum? **Pat Shaughnessy**, 2015. Disponivel em: <http://patshaughnessy.net/2014/1/9/how-big-is-a-bignum>. Acesso em: 24 out. 2015.

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