

**DISCIPLINA: PARADIGMAS DE PROGRAMAÇÃO**

**RELATÓRIO RUBY**

**(2022)**

**ALUNOS:**

* **Jefferson Gomes de Almeida - 192080016;**
* **Rafaela Candido Carneiro Fernandes - 192080393;**
* **Ângelo Gabriel Paz da Silva - 192080075;**
* **Eduarda Fernanda Sousa Cunha - 192080296**
* **Kennedy Johnson de Sousa Dantas - 192080156**

**Prof(ª). Janderson Jason Barbosa Aguiar**

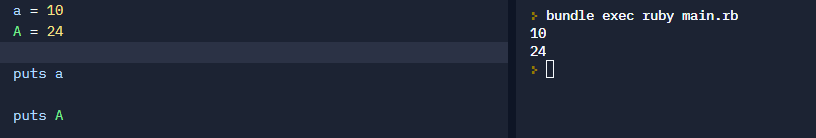
CAMPUS CAMPINA GRANDE, fevereiro de 2022

1. **Considerações iniciais**

O ruby pode ser caracterizado como uma linguagem de alto nível, ou seja, sua sintaxe é voltada para que os humanos a compreendam mais facilmente, de modo intuitivo e amigável para o desenvolvimento e aprendizado.

Criada em 1993 por Yukihiro Matsumoto no japão, ela foi idealizada para ser uma linguagem de script, com o objetivo de balancear a programação funcional e imperativa, se inspirando em linguagens como Python, Smalltalk, Eiffel, Ada e Lib, mas principalmente em python, Ruby é uma linguagem interpretada( Que precisa de um software tradutor para linguagem de máquina), multiparadigma e de tipagem dinâmica forte. Ruby aparece no ranking do Tiobe em 15° lugar no índice de popularidade, e em 16° no IEEE Spectrum, sua popularidade subiu em conjunto com o framework Ruby on Rails.

1. **Conceitos** 
   1. **Nomes**

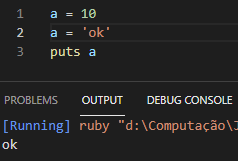
A linguagem Ruby aceita diversos tipos de caracteres, inclusive caracteres especiais como underline e além disso possui distinção entre maiúsculas e minúsculas. 

Quando se trata de tamanho de seus nomes, podem variar de algumas maneiras como o tipo String que tem tamanho máximo: 2\*\*63 - 1, no caso de computadores de 6 bits.

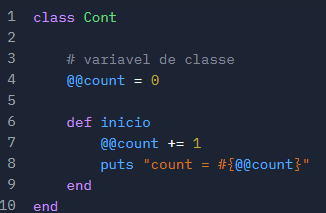
Em se tratando de palavras reservadas pela linguagem, ruby possui por exemplo as palavras reservadas while, BEGIN, return, dentre outras, tornando com que essas palavras não possam ser usadas como variáveis pelo programador.

* 1. **Vinculações (Bindings)**

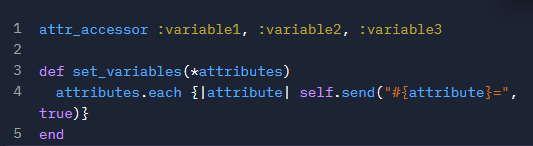
Uma das semelhanças mais importantes que Ruby possui com Python é a vinculação dinâmica, ou seja, o tipo de variável pode ser alterada durante a execução do programa, não é necessária uma sentença de declaração, nem pode ser determinado pelo nome da variável.



O tempo de vida de uma variável é o tempo durante a qual ela está vinculada a uma posição específica da memória. Uma classe pode ter variáveis ​​comuns a todas as instâncias da classe. Essas variáveis ​​são chamadas de variáveis ​​estáticas. Uma variável estática é implementada em ruby ​​usando uma variável de classe. Quando uma variável é declarada como estática, o espaço para ela é alocado durante o tempo de vida do programa.



Nas variáveis dinâmicas de pilha, as vinculações são criadas quando suas sentenças são elaboradas, mas cujo os tipos são estáticamente vinculados. Ruby é uma linguagem altamente dinâmica, no exemplo abaixo é possível ver um exemplo desse comportamento.



Variáveis dinâmicas do monte explícitas são células de memória não nomeadas alocadas e liberadas por instruções explícitas em tempo de execução. Podem ser referenciadas por ponteiros ou variáveis de referência. Ruby não contém ponteiros e não contém variáveis de referência.

Já as variáveis dinâmicas de monte implícitas são vinculadas ao armazenamento no monte apenas quando são atribuídos valores a elas.



A variável meu\_array é um vetor com quatro valores independente de ser usada previamente ou para que foi usada.

* 1. **Escopo**

Uma variável global em Ruby possui um escopo global e pode ser acessada em qualquer parte do programa, elas possuem o símbolo cifrão($) como prefixo e podem ser alteradas em qualquer parte do programa. Já as variáveis locais começam por uma letra minúscula ou por carácter de sublinhado(\_), e assim como as variáveis globais e de instância, não possuem o valor “nil” antes da inicialização.

Essa linguagem possui suas variáveis com escopo determinado antes da execução do programa, e não durante o tempo de execução, como no escopo dinâmico, dito isso, a linguagem Ruby implementa o escopo estático.

* 1. **Tipos de dados**

Em ciência da computação, tipos de dados são uma combinação de valores e de operações que uma variável é capaz de executar, dessa forma, podemos citar que os tipos de dados utilizados pela linguagem Ruby são número básico, String, Ranges, símbolos, verdadeiro, falso, nil e valores especiais, como duas estrutura de dados importantes –Array e Hash. Vale destacar que apesar de definidos assim os tipos em Ruby também são objetos.

Tipos primitivos: número (inteiro, flutuante), string, booleano.

Tipos compostos: classes, arrays (objetos de vários tipos), intervalos, hashes (basicamente um array com um índice de elemento), expressões regulares e símbolos (que no interpretador Ruby representam nomes e alguma string, e é sempre o mesmo objeto durante a execução do programa).

Em Ruby, você não precisa declarar variáveis. Ele apenas usa convenções de nomenclatura para definir o escopo das variáveis.



Para criar uma string basta utilizar aspas, sejam elas simples ou duplas:



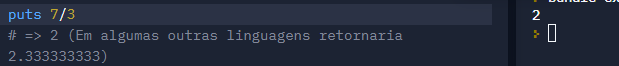
Esta **string** pode ser armazenada em uma **variável** para ser utilizada depois:



Em Ruby string é um tipo primitivo, e tem tamanho dinâmico(limitado), pois embora possa tentar alocar uma string tão grande, ela pode falhar (pelo menos em sistemas de 32 bits, pois normalmente a quantidade máxima de memória que um processo pode alocar é entre 2,5 e 3 GB e 2 \*\* 31 - 1 string em si tem quase 2 GB de comprimento).

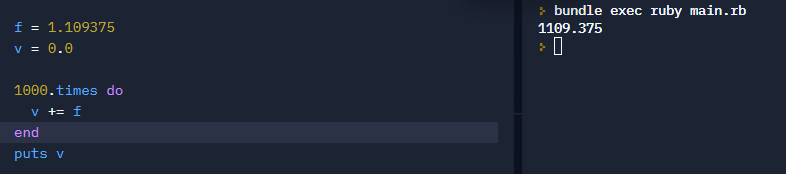
Ruby possui vários tipos numéricos, como: integers, floats, rationals e BigDecimal, sendo integers o mais simples deles.

O Interger é o tipo mais rápido e simples e possui mais funcionalidade em comparação com muitas outras linguagens. Em Ruby, a divisão entre inteiros é tratado de forma similar a linguagem C, onde apenas retorna o quociente e remove todo o resto:

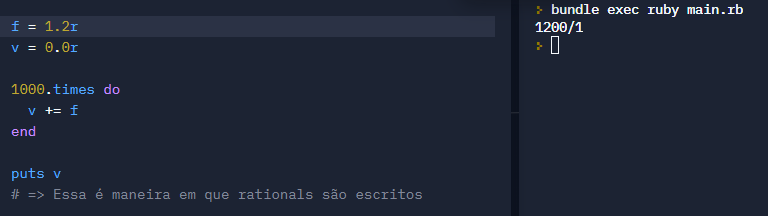


Em casos onde é necessário mostrar a parte fracionada, existem 3 opções de tipos numéricos: Float, Rational e BigDecimal.

O tipo float é o mais rápido das três opções:

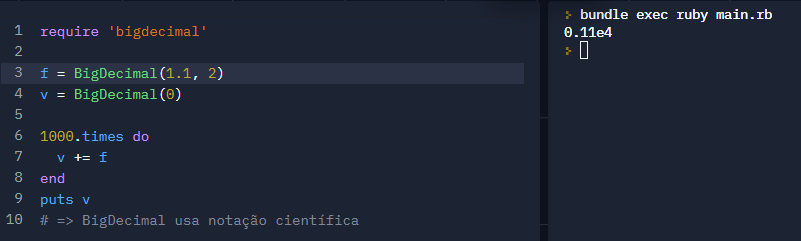


Rational são os mais lentos:

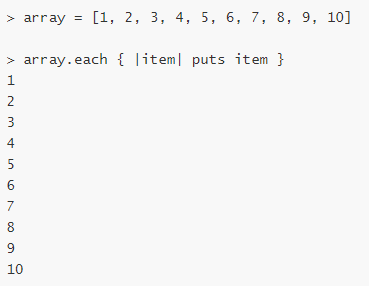


Uma boa regra é usar o Rational sempre que precisar realizar cálculos que precisam de respostas exatas e não estejam entre os inteiros

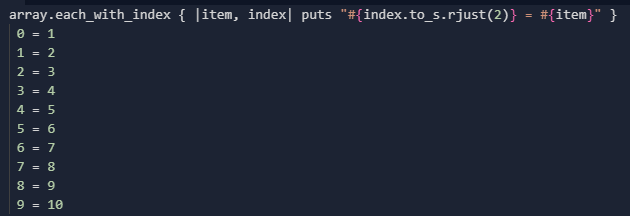
BigDecimal é mais lento que floats e rationals, e por isso, um bom princípio é utilizar BigDecimal apenas em casos em que é necessário lidar com outros sistemas que suportam tipos similares, como tipos de precisão numérica fixas em muitas base de dados, ou quando lidando com outras áreas de precisão fixas, como cálculos monetários.



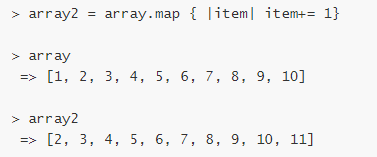
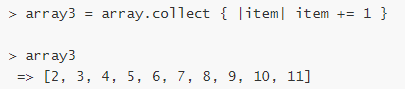
Um tipo ordinal é aquele cuja faixa de valores possíveis pode ser associada ao conjunto dos números inteiros positivos. Podem ser do tipo Enumeração ou do tipo Subfaixa. O primeiro método de enumeração em ruby é o **each,** este método permite percorrer todos os itens de uma coleção e executar para cada item um bloco de código.



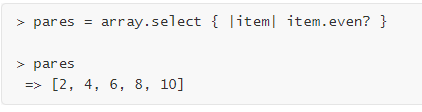
Uma variação do each é a versão com índice, o **each\_with\_index**, que adiciona o índice do array a nossa enumeração.



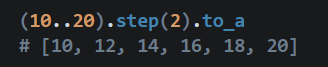
Com o **map** ou **collect** (ambos fazem a mesma coisa) é permitido retornar um novo array com os resultados dos dados manipulados.



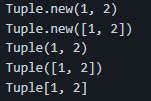
Já com o **select,** é permitidoque executemos uma condição em nosso array, e se ela for verdadeira o item é retornado para o novo array.



A subfaixa é usada para melhorar a legibilidade e confiabilidade do código para o programador, é uma sub sequência contínua de um ordinal, por exemplo, indo de 1 a 20 ou de A até Z. Tipos subfaixas são implementados como seus tipos ancestrais, exceto que as verificações de faixas devem ser implicitamente incluídas pelo compilador em cada atribuição de uma variável.



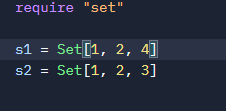
Não existem produtos cartesianos propriamente ditos para ruby, geralmente se utiliza as Structs ou arrays, mas há algumas Gems(gerenciadores de pacotes) uma delas é chamada “tuples”, e nela encontramos duas classes Tuple(No geral) e Pair(Em particular), A principal diferença entre os dois, é que Pair sempre consiste em 2 elementos (Quando inicializado com menos, os ausentes são preenchidos com nil, quando inicializado com mais um ArgumentError é gerado). As formas de inicialização são:



Tanto as tuplas quanto os pares têm métodos auxiliares **length** e **arity** que retorna o número de valores de uma tupla, e **first**,**last** e **second** (Apenas para Pair),

que retornam os respectivos valores, além do método **[]** que dá acesso a valores particulares.

E na gem “sets” teríamos os exemplos:



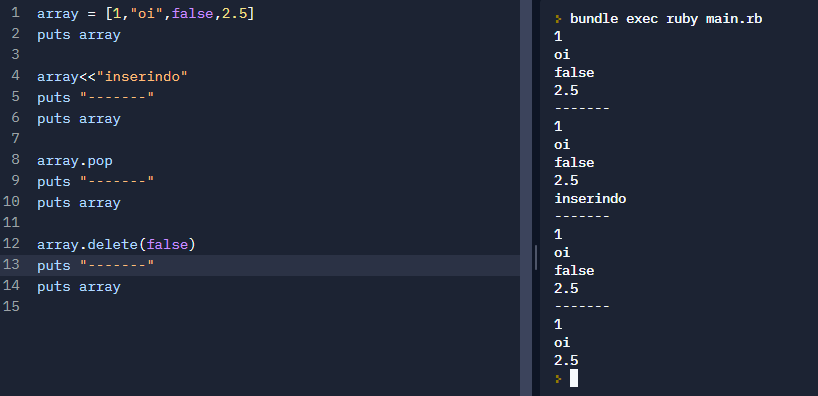
As structs em ruby são classes embutidas, que são usadas para criar novas classes que produzem objetos. Um objeto é usado para armazenar atributos relacionados juntos, um exemplo de criação de uma struct seria:



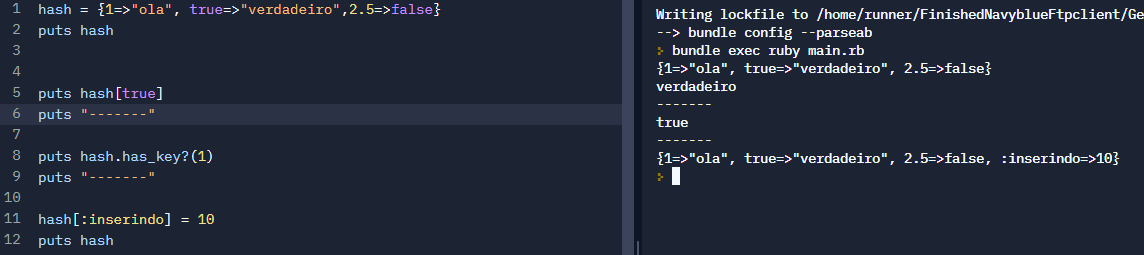
E para retornar os atributos:



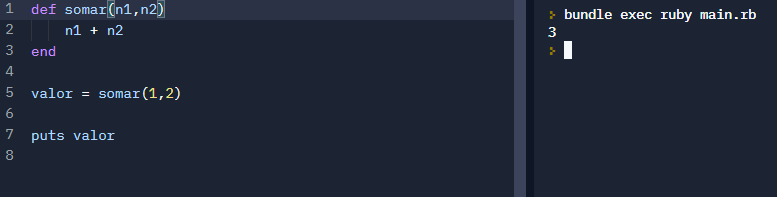
Existem apenas duas estruturas de dados pré-definidas pela linguagem, arrays e hashes. Um array seria o armazenamento de qualquer objeto indexado por inteiros, onde o primeiro valor é zero. Podendo acessá-lo de maneira circular através de índices negativos, sendo alocado dinamicamente na memória, o que significa que irão aumentar de tamanho automaticamente quando forem necessárias novas posições para novos elementos, e possuindo várias funções padrões como: take, drop, push,pop, delete entre outras.



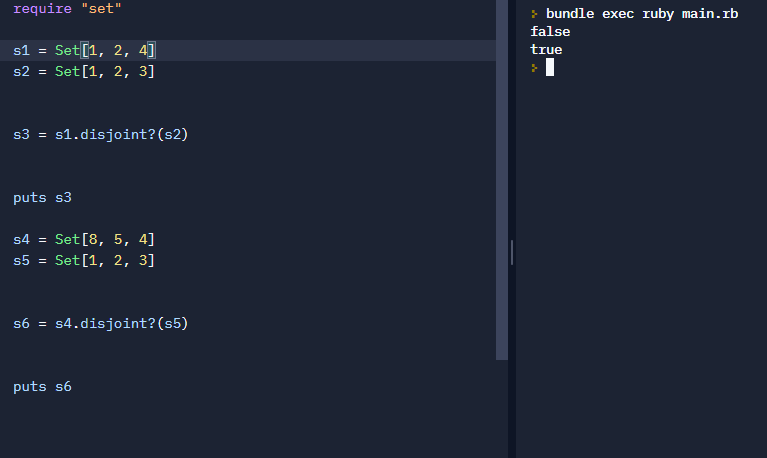
O hash seria uma estrutura semelhante a um array, sendo também dinâmico e também armazenando elementos de diferentes tipos de dados. Mas diferente de um inteiro indexável, seus elementos estão associados a um objeto que funciona como uma chave:



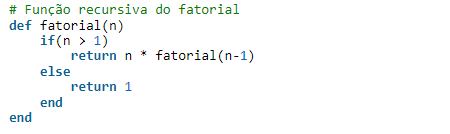
As funções e procedimentos normalmente são descritos como métodos. De forma simples, uma função é um trecho de código que retorna um valor, podendo ser executada várias vezes, e um procedimento executaria um processo em bloco que também pode ser executado várias vezes.



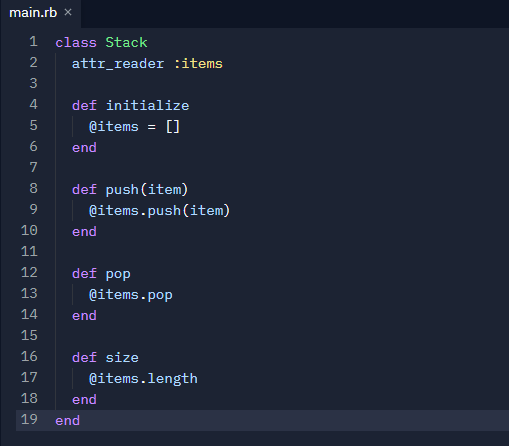
O mais próximo de uma operação de união disjunta que Ruby apresenta seria a função set disjoint, que funciona da seguinte forma ela retorna True se os dois conjuntos apresentados não tiverem nenhum elemento em comum, no caso contrário ele retorna false:



A recursividade é um artifício de programação no qual a definição de uma função ou objeto refere-se ao próprio objeto ou função sendo definido, ou seja chamando a si próprio em seu interior, um exemplo disso seria:



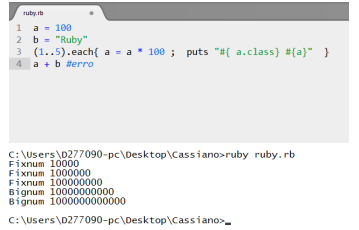
Poderíamos definir uma estrutura dinâmica como uma pilha:



Mas o array e o hash já apresentam a característica de serem dinâmicos, podendo funcionar como pilhas e filas.

* 1. **Checagem de tipos**

Ruby tem tipagem dinâmica, pois a variável pode acomodar objetos de tipos diferentes durante sua vida, além disso também tem tipagem implícita e é fortemente tipada.



* 1. **Equivalência de tipos**

O Ruby possui equivalência estrutural, ou seja, duas variáveis possuem tipos compatíveis se possuem a mesma estrutura. Para se verificar a equivalência estrutural, os nomes dos tipos definidos pelo usuário são substituídos pelas suas definições. Este processo é repetido até não sobrarem mais nomes de tipos definidos pelo usuário.

* 1. **Completude de tipos**

Valores de primeira classe podem ser usados as maneiras possíveis, ou seja,

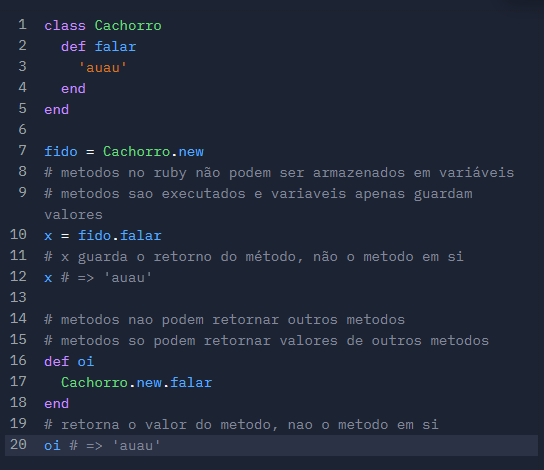
passados como argumentos, avaliados, atribuídos ou fazer parte de valores compostos.

Os métodos em Ruby não são cidadãos de primeira classe porque eles não

podem ser passados para outros métodos como argumentos, retornados por outros

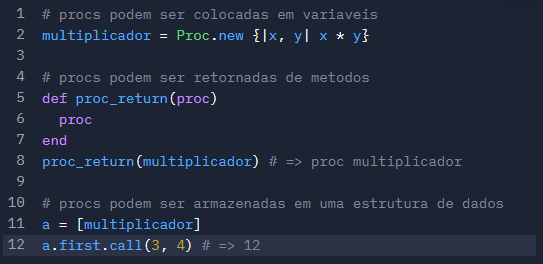
métodos ou colocados em variáveis. Procs no Ruby são primeira classe

semelhantes, similar a funções de primeira classe no JavaScript.



Procs no Ruby são funções anônimas que podem ser tratadas como

cidadãos de primeira classe.



Como Ruby não tem tipos primitivos, e todos os seus tipos são objetos, pode-se

considerar, por exemplo, o tipo Integer como um valor de segunda-classe, pois é uma

classe abstrata e não se pode instanciar objetos com ela.

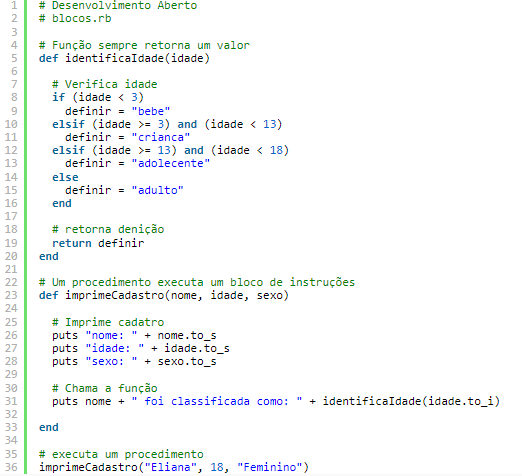
Conforme os exemplos mostrados, é possível afirmar que a linguagem viola o

princípio de completude de tipo.

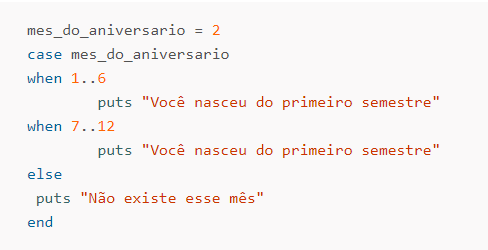
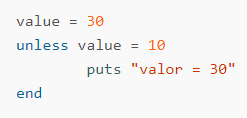
* 1. **Expressões**

Uma expressão em [linguagens de programação](https://pt.wikipedia.org/wiki/Linguagens_de_programa%C3%A7%C3%A3o) é uma combinação de [valores](https://pt.wikipedia.org/wiki/Valor_(computa%C3%A7%C3%A3o)), [variáveis](https://pt.wikipedia.org/wiki/Vari%C3%A1vel_(programa%C3%A7%C3%A3o)), [operadores](https://pt.wikipedia.org/w/index.php?title=Operador_(computa%C3%A7%C3%A3o)&action=edit&redlink=1), e chamadas de [funções](https://pt.wikipedia.org/wiki/Sub-rotina) que são interpretadas de acordo com as [regras de precedência](https://pt.wikipedia.org/wiki/Ordem_de_opera%C3%A7%C3%B5es) e de [associatividade](https://pt.wikipedia.org/wiki/Associatividade_de_operadores) particulares a uma determinada linguagem de programação, que calcula e, em seguida, produz um valor. Expressões são os meios fundamentais de especificação de computações em uma linguagem de programação. O propósito de uma expressão é especificar um valor a ser calculado.

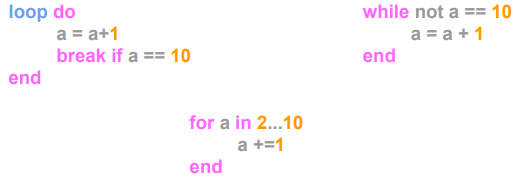
Em ruby, suas expressões literais são booleanos e zero, números, Strings, símbolos, arrays, hashes, ranges, expressões regulares e procs. As funções e procedimentos são geralmente referidos como métodos. Basicamente uma função retorna um valor e um procedimento executa um processo em bloco.



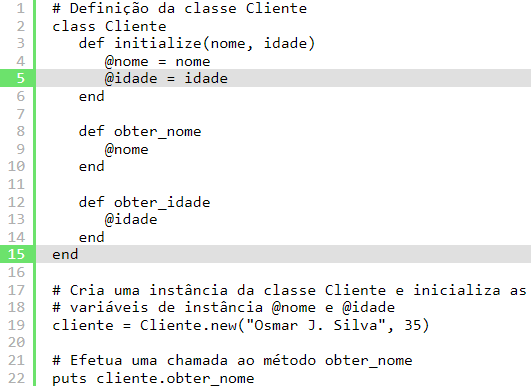
As expressões condicionais são **If**, **else**, sua junção chamada **elsif**, **unless** e **case**.



Já as iterativas são o **do-while**, **while** e **for**.



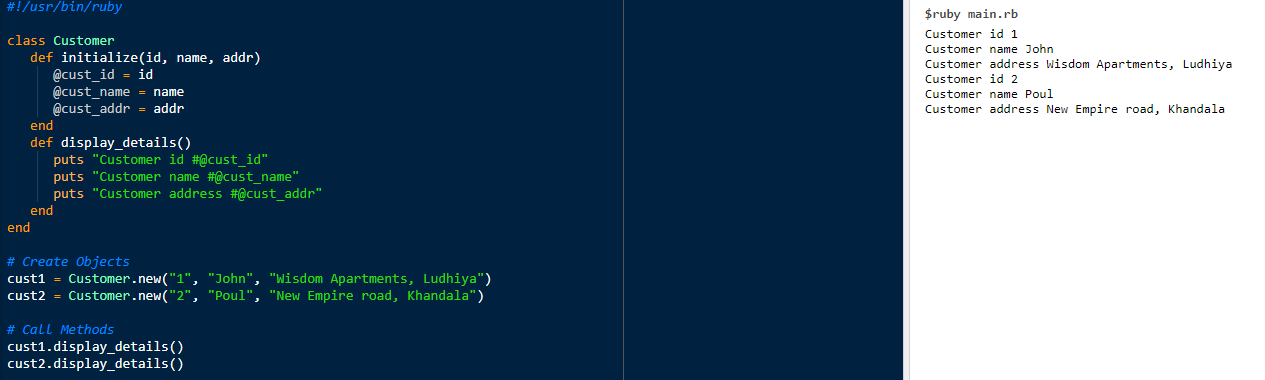
Os construtores(agregados) são recursos muito importantes na construção de objetos de um classe, uma vez que estes permitem inicializar as variáveis de instância do objeto sendo construído. Em Ruby, o construtor de uma classe é definido com o uso da palavra-chave initialize.



As constantes devem começar com letra maiúscula. Constantes definidas dentro de uma classe ou módulo podem ser acessadas de dentro dessa classe ou módulo, e aquelas definidas fora de uma classe ou módulo podem ser acessadas globalmente.



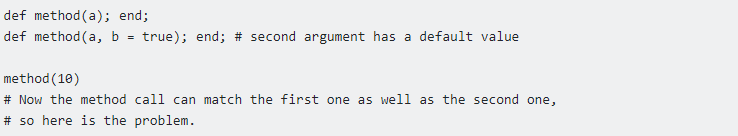
As variáveis ​​de instância começam com @. As variáveis ​​de instância não inicializadas têm o valor *nil* e produzem avisos com a opção -w.



* 1. **Sistemas de tipos**

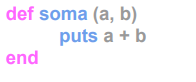
O conceito de sistemas de tipos pode ser descrito como uma definição precisa das associações entre o tipo de uma variável, seus valores e as operações possíveis com esses valores, posto isso podemos observar algumas características de ruby relacionadas a isso.

Em Ruby não existe sobrecarga, mesmo que dois métodos com o mesmo nome tenham parâmetros diferentes:

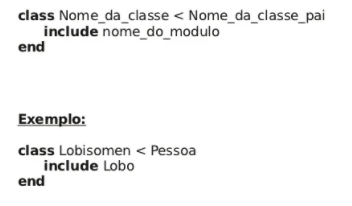


Portanto o Ruby precisa manter um método na cadeia de pesquisa com um nome exclusivo.

Os métodos em Ruby são naturalmente paramétricos ou seja não é necessário declarar os tipos dos seus parâmetros:



Graças às heranças executadas em ruby é possível utilizar o polimorfismo de inclusão, mas deve-se ter precaução ao utilizá-lo, visto que não há encapsulamento entre objetos.



O polimorfismo por coerção ocorre quando um tipo primitivo ou um objeto é convertido em outro tipo primitivo ou de objeto, e o ruby possui polimorfismo por coerção apesar de às vezes poder causar erros sutis no programa.

