2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e Ruby

Índice

[Nota Geral: 3](#_Toc358236426)

[Algumas notas sobre Ruby: 3](#_Toc358236427)

[Estrutura Início 4](#_Toc358236428)

[Início: 4](#_Toc358236429)

[Estrutura Fim 4](#_Toc358236430)

[Fim: 4](#_Toc358236431)

[Variáveis 4](#_Toc358236432)

[Tipos de variáveis 4](#_Toc358236433)

[Definição e atribuição de variáveis 5](#_Toc358236434)

[Estruturas input/output 6](#_Toc358236435)

[Input – Ler 6](#_Toc358236436)

[Output – Escrever 6](#_Toc358236437)

[Estruturas de Decisão 7](#_Toc358236438)

[Condição *“if”* 7](#_Toc358236439)

[Condição *“if/else”* 7](#_Toc358236440)

[Condição *“else if”* 8](#_Toc358236441)

[Condição *“while”* 8](#_Toc358236442)

[Condição *“do while”* 9](#_Toc358236443)

[Funções 9](#_Toc358236444)

[Definir funções 9](#_Toc358236445)

[Chamada de funções 10](#_Toc358236446)

[Estrutura de retorno 10](#_Toc358236447)

[Return 10](#_Toc358236448)

[Operadores 11](#_Toc358236449)

[Aritméticos 11](#_Toc358236450)

[Lógicos 11](#_Toc358236451)

[Relacionais 11](#_Toc358236452)

[ANEXO 12](#_Toc358236453)

[Algoritmo com o uso da condição “if” 12](#_Toc358236454)

[Fluxograma 12](#_Toc358236455)

[Código 12](#_Toc358236456)

[Algoritmo com o uso da condição “if else” 13](#_Toc358236457)

[Fluxograma 13](#_Toc358236458)

[Código: 13](#_Toc358236459)

[Algoritmo com o uso da condição “while” 14](#_Toc358236460)

[Fluxograma 14](#_Toc358236461)

[Código 14](#_Toc358236462)

[Algoritmo com o uso da condição “ do while” 15](#_Toc358236463)

[Fluxograma 15](#_Toc358236464)

[Código 15](#_Toc358236465)

[Algoritmo com o uso de uma função 16](#_Toc358236466)

[Fluxogramas 16](#_Toc358236467)

[Código 16](#_Toc358236468)

# Nota Geral:

Devido à especificação da linguagem, a tradução só é possível depois de ser executado o fluxograma.

# Algumas notas sobre Ruby:

* Lançado em 1995, no Japão, por [Yukihiro "Matz" Matsumoto](http://pt.wikipedia.org/wiki/Yukihiro_Matsumoto" \o "Yukihiro Matsumoto);
* Linguagem de *script*, orientada a objectos (todas as variáveis são objectos, tudo tem um valor);
* Suporta apenas herança simples e *multithreading*;
* Multiplataforma;
* Os comentários do código, são iniciados pelo caracter cardinal (#);
* O código deve ser guardado num ficheiro com extensão .rb .

# Estrutura Início

## Início:

Início

Apenas em ambiente Linux é necessário iniciar o programa com a instrução #!/usr/bin/ruby  
por forma a identificar a bash/shell (interpretador de comandos) que vai ser usado.  
Noutras plataformas não é necessário nem usada nenhuma instrução específica para indicar o começo do programa.

# Estrutura Fim

## Fim:

Fim

Não é necessário nem usada nenhuma instrução específica para indicar o final do programa.

# Variáveis

## Tipos de variáveis

Em *Ruby* as variáveis não ficam restringidas a ter um único tipo de dados, o interpretador irá automaticamente definir o seu tipo baseado no contexto em que a variável se associa.  
Tendo em conta que tudo são objectos, os mais habitualmente utilizados são:

* **Numéricos:** dividem-se entre inteiro (integer) e decimal (float);
* **Strings:** texto ou conjunto de caracteres, delimitados por plicas (‘) ou aspas (“);
* **Arrays:** representam matrizes e vectores, delimitados por parêntesis rectos ([]) e cada valor separado por vírgula;
* **Hashes:** representam vectores associativos, delimitados por chavetas ({}) e o índice precede o valor com um sinal '=>' ;
* **Regexp:** representam expressões regulares, delimitadas por //.

## Definição e atribuição de variáveis

variavel <- valor

Os nomes das variáveis locais devem sempre iniciar-se com letra minúscula ou com o caracter *underscore* (\_);

Se uma variável começar com:

* Letra Maiúscula, é uma *constante*;
* $, é uma variável global;
* @, é uma variável de instância (atributo de um objecto);
* @@ é uma variável de classe.
* **Constante**
  + **Ex:** CONST = 3,1415
* **Variável Global**
  + **Ex:** $variavel\_global = “abc”
* **Variável de Instância**

Estas variáveis devem ser inicializadas e usadas nos métodos de instância. Quando definidas no corpo da classe, ficam no contexto da classe:

**Ex:**

class A

@contexto = "classe"

def initialize

@contexto = "instância"

end

def contexto

@contexto

end

def A.contexto

@contexto

end

end

a = A.new

a.contexto **# Devolverá: "instância"**

A.contexto **# Devolverá: "classe"**

* **Variável de Classe**

**Ex:**

class Carro

@@marcas = [ "Ford", "GM", "Fiat", "VW" ]

end

# Estruturas input/output

## Input – Ler

variavel

Para ler do teclado, teremos que atribuir a instrução gets à variável que vai estar associado o valor introduzido:  
  
num = ***gets***

## Output – Escrever

expressao

Para escrever no ecrã, são utilizadas duas instruções: ***puts*** e ***print.***Com a instrução ***puts***, é sempre assumida uma quebra de linha no final do output, enquanto que na instrução ***print***, teríamos de adicionar **\n** para que isso acontecesse.

**Ex:**

print “Nome”  
print “Apelido”  
  
Teremos no ecrã:   
NomeApelido

Mas se colocar-mos:  
puts “Nome”  
puts “Apelido”  
  
Teremos:   
Nome  
Apelido

# Estruturas de Decisão

## Condição *“if”*

condição

true

if (*condição*)   
Instruções  
end

**Nota**: A condição só necessita de estar dentro de parêntesis se tiver mais que um operador

## Condição *“if/else”*

condição

false

true

if (*condição*)  
 Instruções  
else  
Instruções  
end

**Nota**: A condição só necessita de estar dentro de parêntesis se tiver mais que um operador

## Condição *“else if”*

if (*condição1*)  
Instruções  
elsif(condição2)  
Instruções  
else  
Instruções  
end

**Nota**: A condição só necessita de estar dentro de parêntesis se tiver mais que um operador

## Condição *“while”*

condição

true

false

instruções

while (*condição*)

Instruções

end

**Nota**: A condição só necessita de estar dentro de parêntesis se tiver mais que um operador

## Condição *“do while”*

instruções

TRUE

condição

FALSE

until (condição)

Instruções

end

**Nota**: A condição só necessita de estar dentro de parêntesis se tiver mais que um operador

# Funções

## Definir funções

As funções são habitualmente definidas na parte inicial do programa, antes das instruções que farão a execução principal do programa. São habitualmente definidas por:

**def** nomeFuncao1

Instruções

**end**

Quando existe necessidade de definir uma função com parâmetros:  
 **def** nomeFuncao2 variavel  
 Instruções  
**end**

## Chamada de funções

Aproveitando o exemplo das funções acima declaradas:  
  
nomeFuncao1 #Chamada da 1ª função

x = nomeFuncao2 500

puts x #Chamada da 2ª função, tendo como parâmetro o valor 500

# Estrutura de retorno

## Return

expressao

É definido dentro da função implementada:

return expressão

**Ex:**

def test

i = 100

j = 200

k = 300

**return** i, j, k

end

var = test

puts var

# Operadores

### Aritméticos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Ruby |
| Adição | a + b | a + b |
| Subtração | a – b | a - b |
| Divisão | a / b | a / b |
| Multiplicação | a \* b | a \* b |
| Resto da divisão inteira |  | a % b |
| Potenciação |  | \*\* |
| Concatenação de texto |  | + |

Tabela 1 - Equivalência de operadores aritméticos

### Lógicos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Ruby |
| Disjunção | a && b | a && b |
| Conjunção | a || b | a || b |
| Conjunção Exclusiva | a ^ b | a ^ b |
| Negação |  | a !b |

Tabela 2- Equivalência de operadores lógicos

### Relacionais

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Ruby |
| Igual | a == b | a == b |
| Diferente | a != b | a != b |
| Maior | a > b | a > b |
| Maior ou igual | a >= b | a >= b |
| Menor | a < b | a < b |
| Menor ou igual | a <= b | a <= b |

Tabela 3- Equivalência de operadores relacionais

# ***ANEXO***

Para uma compreensão mais abrangente do uso das estruturas, ficam alguns exemplos mais extensivos, com o uso de várias estruturas em algoritmos completos.

## Algoritmo com o uso da condição “if”

**Problema:** Verificar se um número introduzido pelo utilizador é par.

### Fluxograma

Inicio

n

n%2=0

TRUE

FALSE

“Par”

Fim

### Código

### 

## Algoritmo com o uso da condição “if else”

**Problema:** Verificar se um número introduzido pelo utilizador é par ou ímpar.

### Fluxograma

### 

TRUE

Fim

“Par”

n%2=0

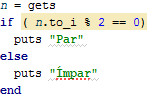
n

Inicio

FALSE

“Impar”

### Código:



## Algoritmo com o uso da condição “while”

**Problema:** Escrever um número de 1 a 10.

### Fluxograma

Inicio

i=1

i <= 10

FALSE

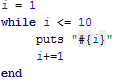
TRUE

i

i=i + 1

Fim

### Código



## Algoritmo com o uso da condição “ do while”

**Problema:** Pedir um número positivo.

### Fluxograma

Inicio

i = 0

i

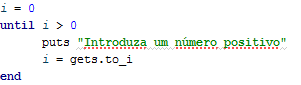
TRUE

i < 0

FALSE

Fim

### Código



## Algoritmo com o uso de uma função

**Problema:** Factorial de um número.

### Fluxogramas

**Código principal**

**Função fact(k)**

fact(k)

Inicio

TRUE

FALSE

k > 2

i = 5

j = fact ( i )

k

k \* fact(k - 1)

j

Fim

Fim

### Código

