

Introdução   
 ao   
 Ruby

Fellipe   
 Aleixo   
 (*fellipe.aleixo@ifrn.edu.br*)

|  |  |
| --- | --- |
| Ruby |  |

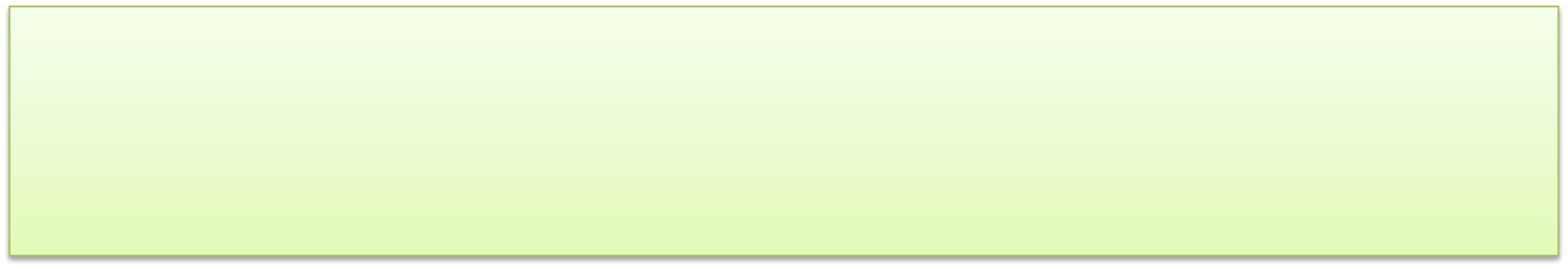
•Linguagem   
 orientada   
 a   
 objetos   
 –Tudo   
 o   
 que   
 você   
 manipula   
 em   
 Ruby   
 é   
 um   
 objeto –O   
 resultado   
 dessas   
 manipulações   
 também   
 são objetos

•Modelagem   
 orientada   
 a   
 objetos   
 à   
 modelar conceitos   
 do   
 “mundo   
 real”   
 –Classes   
 à   
 das   
 quais   
 são   
 gerados   
 os   
 objetos

Orientação   
 a   
 Objetos •Um   
 objeto   
 é   
 a   
 combinação   
 de   
 –Estado   
 (atributos   
 e   
 seus   
 valores)   
 –Métodos   
 (operam   
 sobre   
 o   
 estado) 

•São   
 criados   
 por   
 meio   
 de   
 um   
 construtor –Construtor   
 padrão   
 =   
 **new()**

|  |
| --- |
| line\_item\_one   =   LineItem.new  line\_item\_one.quanPty   =   1  line\_item\_one.sku   =   "AUTO\_B\_00" |

Métodos   
•Métodos   
 são   
 invocados   
 pelo   
 envio   
 de   
 uma mensagem   
 para   
 um   
 objeto   
 –Mensagem   
 =   
 nome   
 do   
 método   
 [+   
 parâmetros] 

|  |
| --- |
| "dave".length  line\_item\_one.quanPty()  cart.add\_line\_item(next\_purchase)  submit\_tag   "Add   to   Cart" |

–Os   
 parênteses   
 são,   
 geralmente,   
 opcionais   
 na chamada   
 dos   
 métodos

Nomenclatura

•(i)   
 variáveis   
 locais,   
 (ii)   
 parâmetros   
 de   
 método e   
 (iii)   
 nomes   
 de   
 métodos   
 devem   
 iniciar   
 com uma   
 letra   
 minúscula   
 ou   
 um   
 “underline”   
 (\_)

–Exemplos:   
 order,   
 line\_item   
 e   
 xr2000

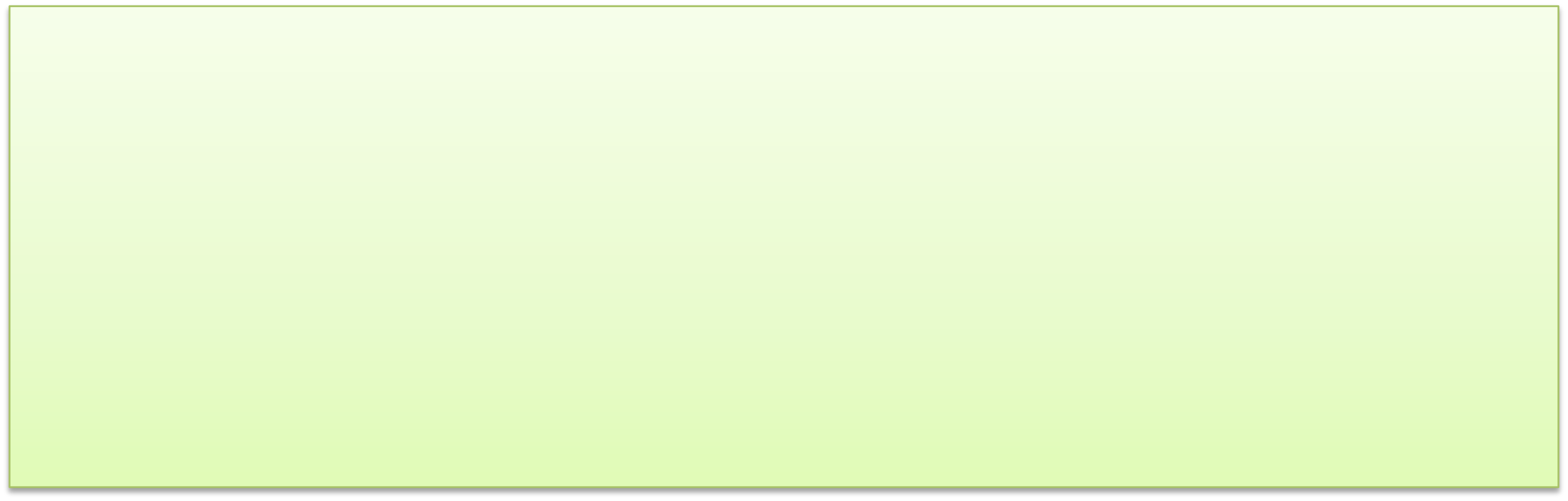
•Atributos   
 (variáveis   
 de   
 instância)   
 devem   
 iniciar com   
 um   
 arroba   
 (@)

–Exemplos:   
 @quanPty   
 e   
 @product\_id

–“underline”   
 é   
 usado   
 para   
 unir   
 várias   
 palavras

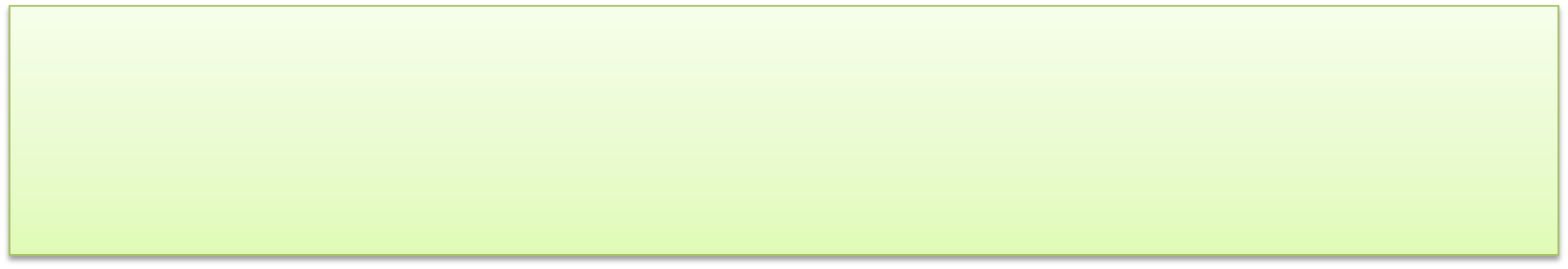
Nomenclatura   
•(i)   
 classes,   
 (ii)   
 módulos   
 e   
 (iii)   
 constantes devem   
 iniciar   
 com   
 uma   
 letra   
 maiúscula –União   
 de   
 várias   
 palavras   
 com   
 letras   
 maiúsculas –Exemplos:   
 Object,   
 PurchaseOrder   
 e   
 LineItem •Rails   
 uPliza   
 símbolos   
 para   
 idenPficar   
 coisas –Nomeando   
 parâmetros   
 de   
 métodos 

|  |
| --- |
| redirect\_to   :acPon   =>   "edit",   :id   =>   params[:id] |

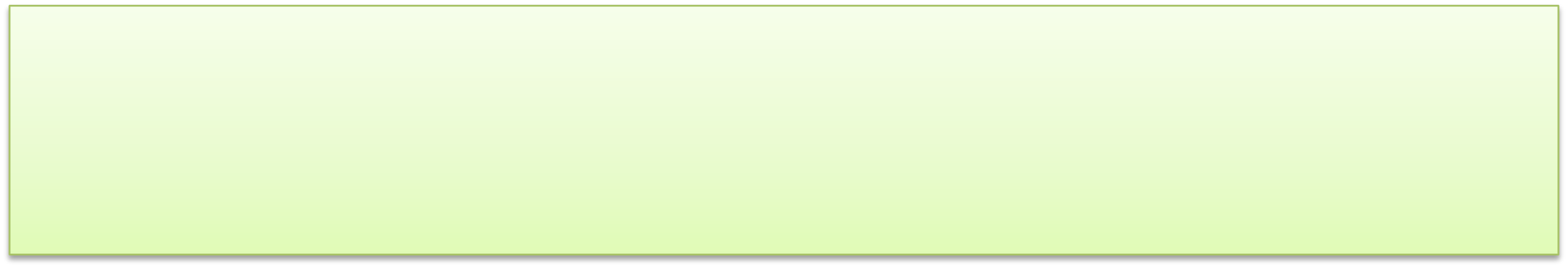
Métodos   
•Definição   
 de   
 funções   
 que,   
 opcionalmente,   
 (i) recebem   
 parâmetros   
 e   
 (ii)   
 geram   
 resultados 

|  |
| --- |
| def   say\_goodnight(name)           result   =   'Good   night,   '   +   name           return   result  end  #   Time   for   bed...  puts   say\_goodnight('Mary-­‐Ellen')                         #   =>   'Goodnight,   Mary-­‐Ellen' puts   say\_goodnight('John-­‐Boy')                               #   =>   'Goodnight,   John-­‐Boy' |

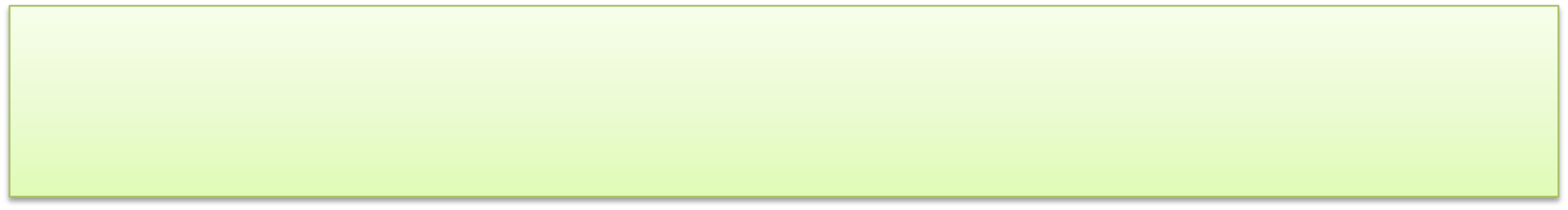
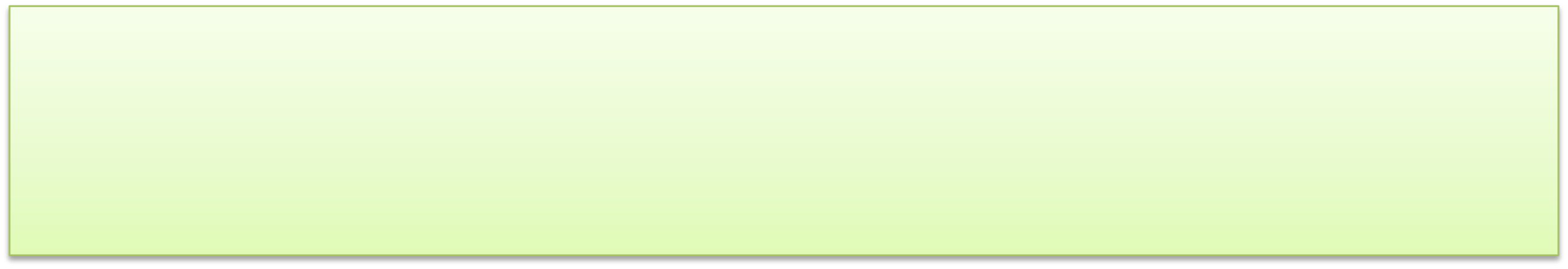
–Caracter   
 “#”   
 defini   
 um   
 comentário

Tipos   
 de   
 Dados   
•***Strings***   
 –Pode   
 ser   
 criado   
 a   
 parPr   
 de   
 *strings*   
 literais   
 •Separados   
 por   
 aspas   
 simples   
 (‘)   
 ou   
 duplas   
 (“)   
 •Para   
 os   
 *strings*   
 entre   
 aspas   
 duplas   
 –   
 Ruby   
 trabalha   
 um pouco   
 mais   
 –Primeiro   
 procura   
 por   
 subsPtuições   
 (Ex.:   
 \n)   
 –Depois   
 faz   
 a   
 interpretação   
 de   
 expressões   
 (#{expressão}) 

|  |
| --- |
| def   say\_goodnight(name)           "Good   night,   #{name.capitalize}"  end  puts   say\_goodnight('pa') |

*Arrays*   
 e   
 *Hashes*   
•Possibilitam   
 trabalhar   
 com   
 coleções   
 de   
 objetos   
 indexados   
 (acessados   
 por   
 um   
 chave) –Nos   
 *arrays*   
 a   
 chave   
 é   
 um   
 número   
 inteiro   
 –Nos   
 *hashes*   
 a   
 chave   
 é   
 um   
 objeto   
 qualquer   
•Crescem   
 para   
 armazenar   
 novos   
 elementos 

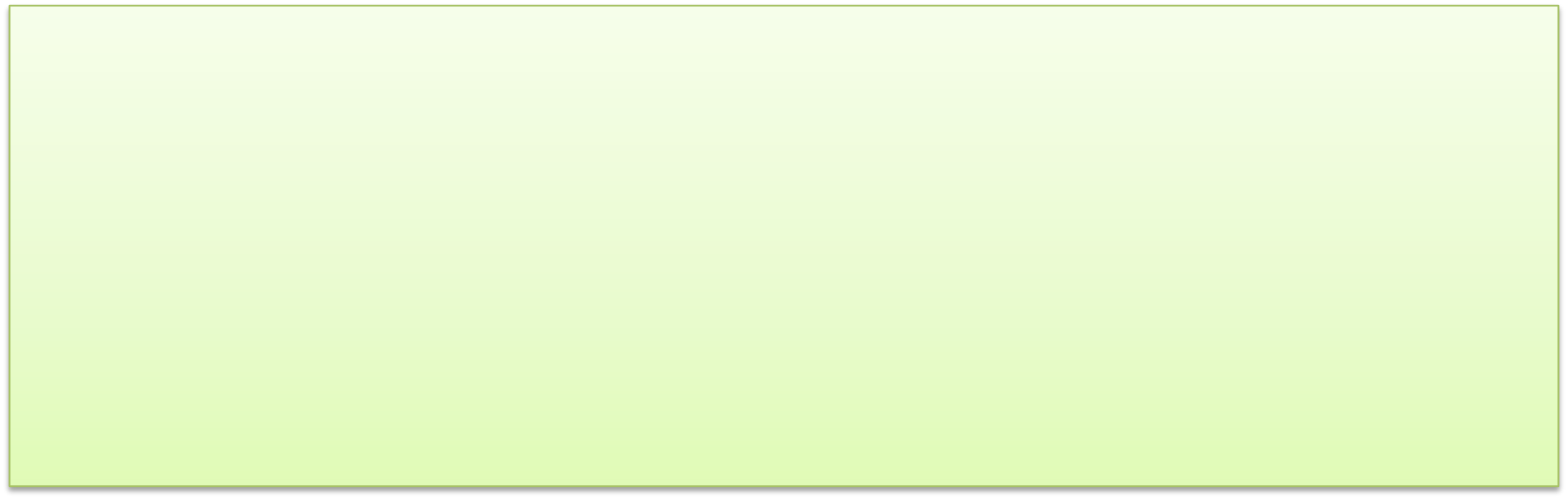
|  |
| --- |
| a   =   [   1,   'cat',   3.14   ]                         #   array   with   three   elements a[0]                                                                           #   access   the   first   element   (1) a[2]   =   nil                                                         #   set   the   third   element                                                                                         #   array   now   [   1,   'cat',   nil   ] |

*Arrays*   
 e   
 *Hashes*  
•O   
 método   
 “<<()”   
 é   
 uPlizado   
 com   
 *arrays* 

|  |
| --- |
| ages   =   []  for   person   in   @people           ages   <<   person.age  end |

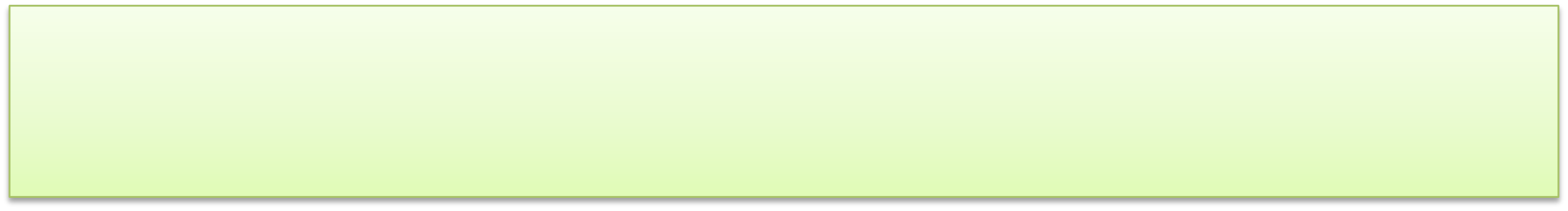
•Atalhos   
 para   
 a   
 criação   
 de   
 *arrays*

|  |
| --- |
| a   =   [   'ant',   'bee',   'cat',   'dog',   'elk'   ]  #   this   is   the   same:  a   =   %w{   ant   bee   cat   dog   elk   } |

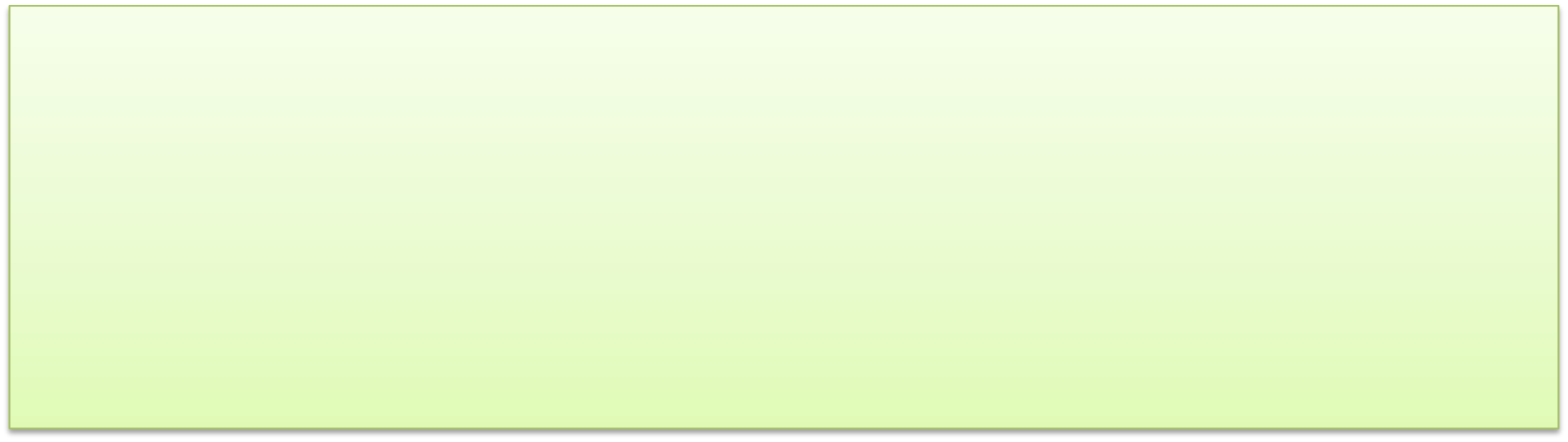
*Arrays*   
 e   
 *Hashes*  
•Nos   
 *hashes*   
 são   
 uPlizadas   
 chaves   
 para   
 a inserção   
 e   
 recuperação   
 de   
 elementos 

|  |
| --- |
| inst\_secPon   =   {           :cello   =>   'string',           :clarinet   =>   'woodwind',           :drum   =>   'percussion',           :oboe   =>   'woodwind',           :trumpet   =>   'brass',           :violin   =>   'string'  } |

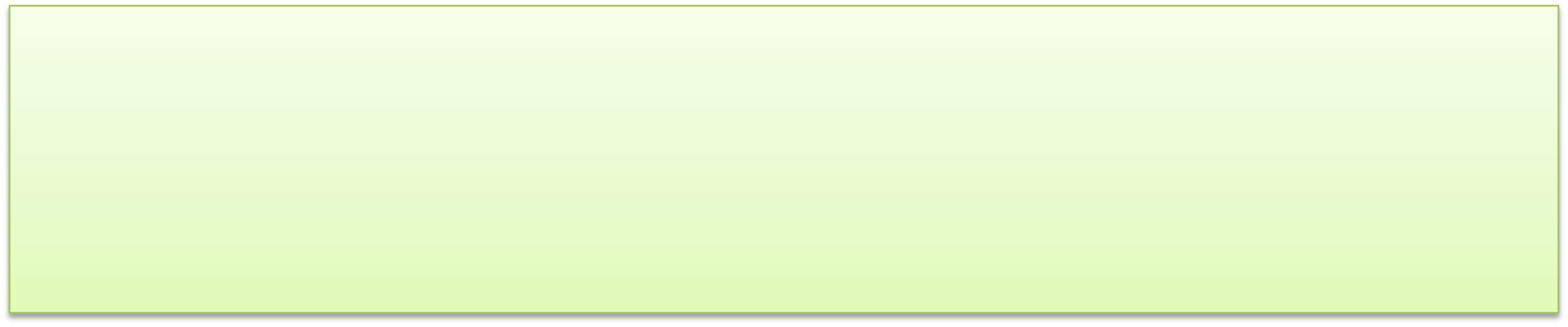
–“=>”   
 é   
 opcional

Expressões   
 Regulares •Uma   
 expressão   
 regular   
 permite   
 a especificação   
 padrão   
 de   
 caracteres –Para   
 localizar   
 a   
 mesma   
 em   
 uma   
 *string*  –Criado   
 com   
 **/pa\*ern/**   
 ou   
 **%r{pa\*ern}** 

|  |
| --- |
| if   line   =~   /P(erl|ython)/           puts   "There   seems   to   be   another   scripPng   language   here" end |

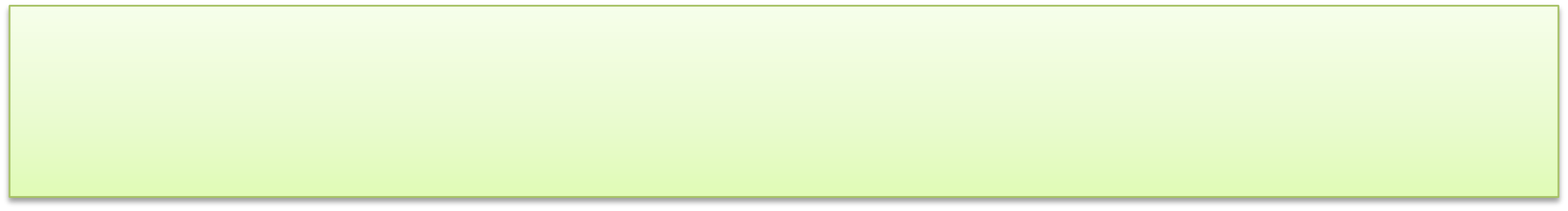
Estruturas   
 de   
 Controle   
 de   
 Fluxo •Instrução   
 **if** 

|  |
| --- |
| if   count   >   10           puts   "Try   again"  elsif   tries   ==   3           puts   "You   lose"  else           puts   "Enter   a   number"  end |

Estruturas   
 de   
 Controle   
 de   
 Fluxo •Instrução   
 **while** 

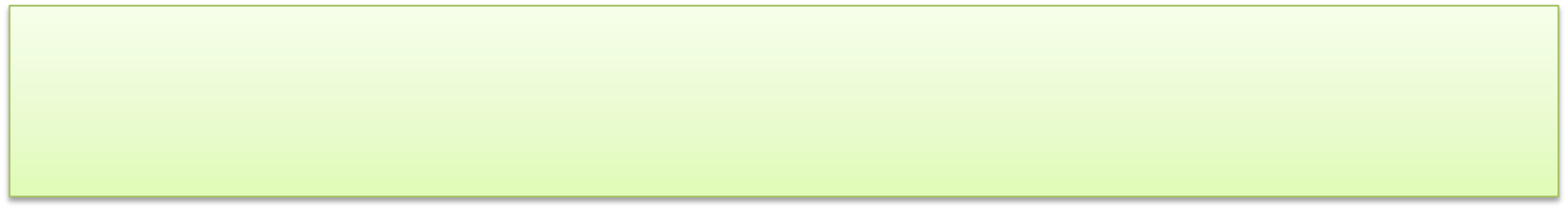
|  |
| --- |
| while   weight   <   100   and   num\_pallets   <=   30         pallet   =   next\_pallet()           weight   +=   pallet.weight           num\_pallets   +=   1  end |

•Algumas   
 variantes   
 precisam   
 de   
 atenção –Ex.:   
 **unless**   
 e   
 **un5l**   
–Expressões   
 em   
 uma   
 única   
 linha

Estruturas   
 de   
 Controle   
 de   
 Fluxo •Expressões   
 em   
 uma   
 linha

|  |
| --- |
| puts   "Danger,   Will   Robinson"   if   radiaPon   >   3000 distance   =   distance   \*   1.2   while   distance   <   100 |

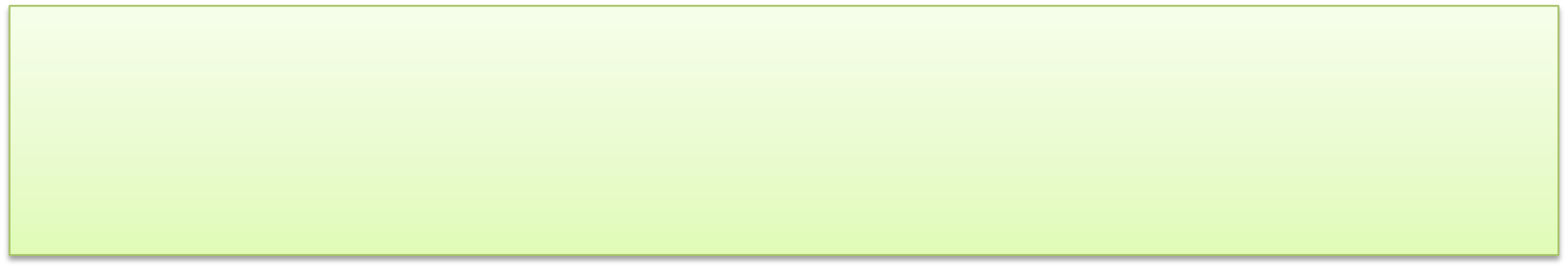
•Algumas   
 variantes   
 precisam   
 de   
 atenção –Ex.:   
 **unless**   
 e   
 **un5l**   
–Expressões   
 em   
 uma   
 única   
 linha

Blocos   
 e   
 Iteradores •Blocos   
 de   
 código 

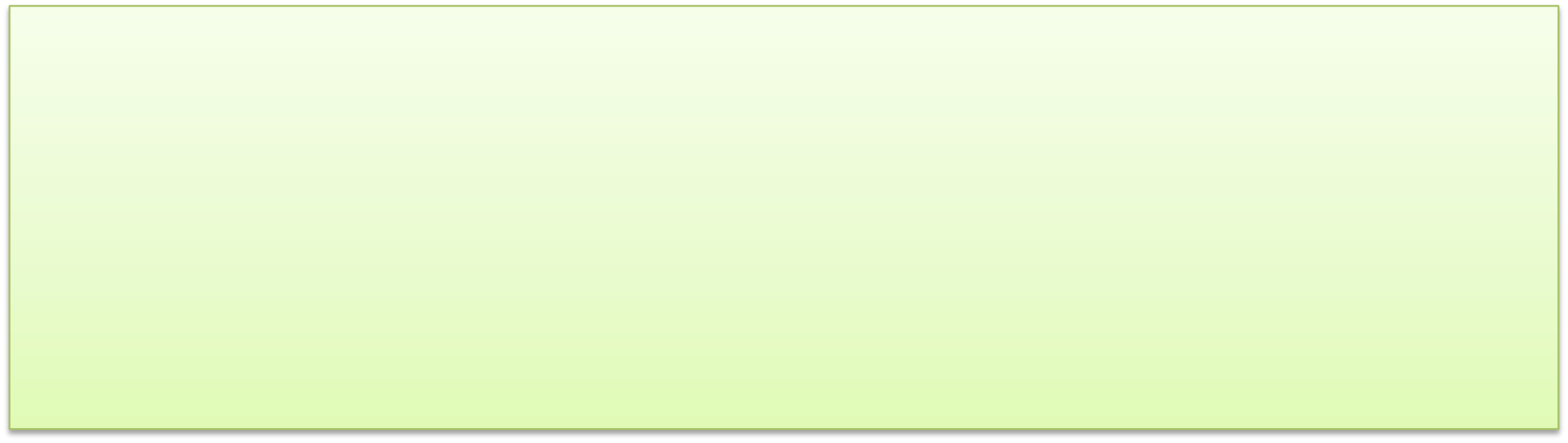
|  |
| --- |
| {   puts   "Hello"   }                                     #   uPlizado   geralmente   em   blocos   de   uma   linha  do                                                                                 ###           club.enroll(person)                       #   também   representa   um   bloco         person.socialize                                 #  end                                                                           ### |

–Pode   
 ser   
 passado   
 um   
 bloco   
 para   
 um   
 método

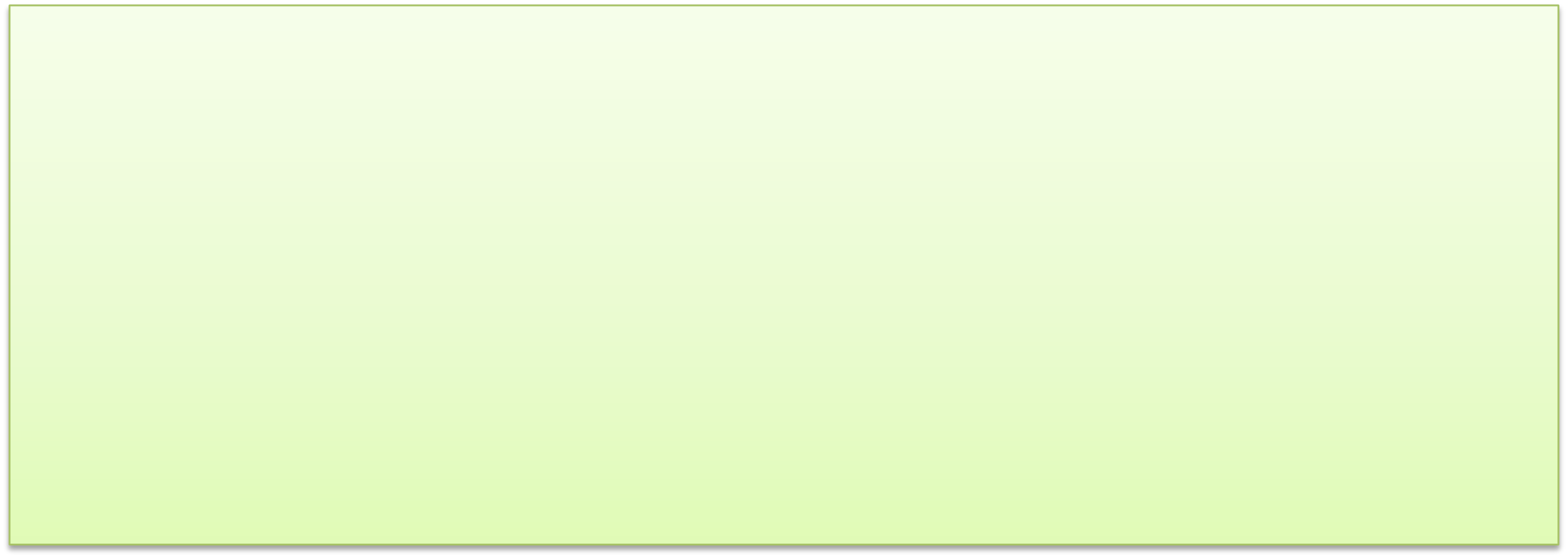
|  |
| --- |
| greet   {   puts   "Hi"   }  #   passando   também   parâmetros  verbose\_greet("Dave",   "loyal   customer")   {   puts   "Hi"   } |

Instrução   
 **yield**   
•Chamada   
 de   
 um   
 bloco   
 associado   
 –Podem   
 ser   
 passados   
 parâmetros   
 para   
 o   
 bloco   
 – nominados   
 entre   
 barras   
 verPcais   
 (|)   
•Blocos   
 são   
 comumente   
 usados   
 com   
 iteradores 

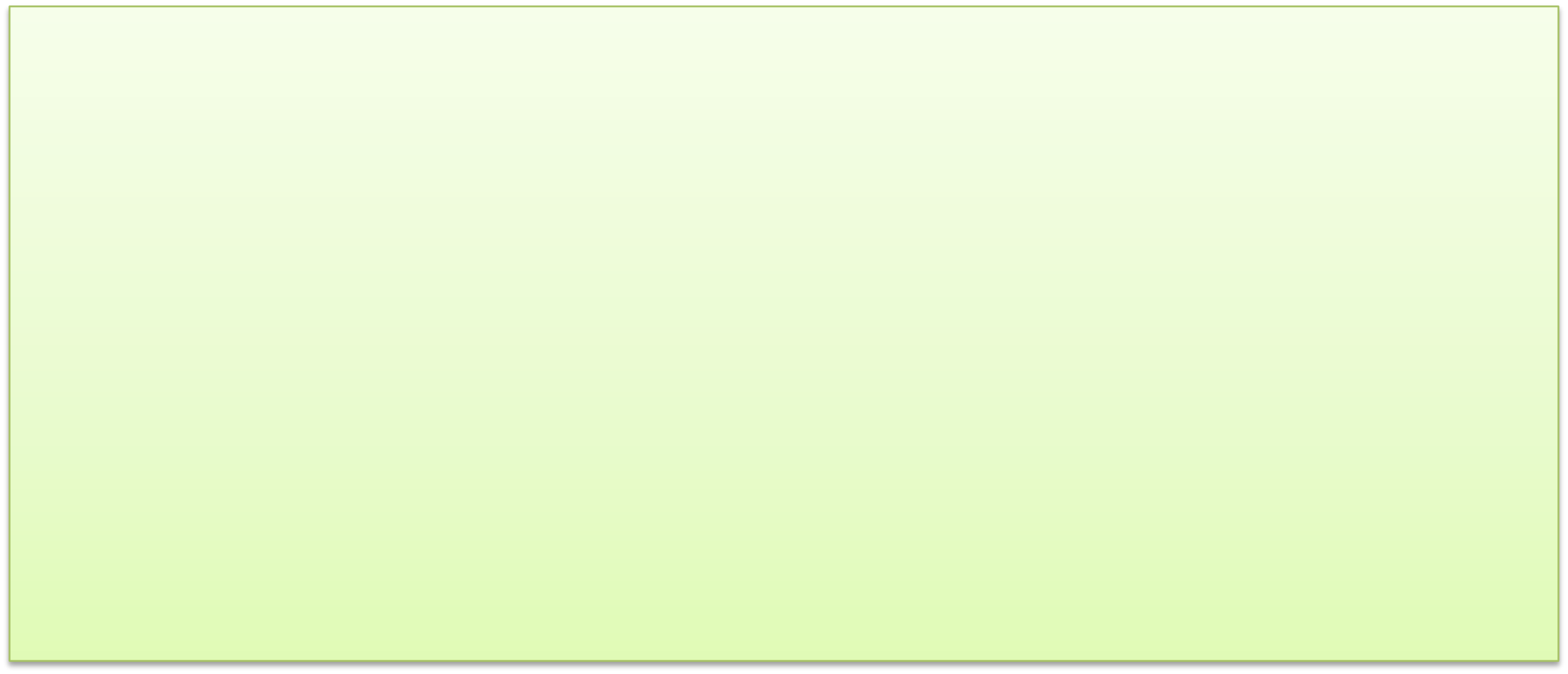
|  |
| --- |
| animals   =   %w(   ant   bee   cat   dog   elk   )                         #   cria   um   array  animals.each   {|animal|   puts   animal   }                 #   itera   no   seu   conteúdo  3.Pmes   {   print   "Ho!   "   }                                                                     #=>   Ho!   Ho!   Ho! |

Instrução   
 **yield**  
•Capturando   
 e   
 nomeando   
 um   
 bloco   
 passado para   
 um   
 método 

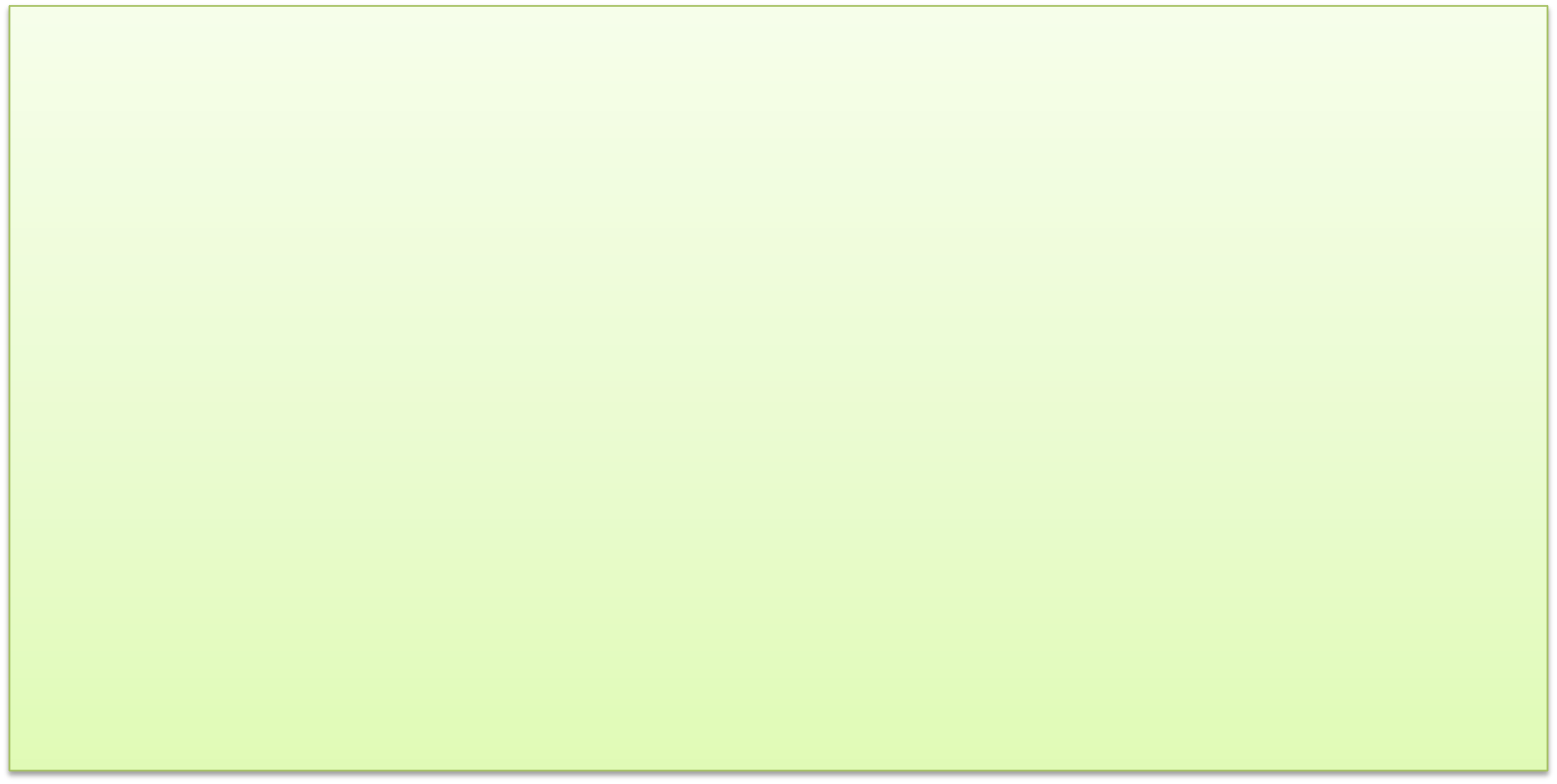
|  |
| --- |
| def   wrap   &b           print   "Santa   says:   "           3.Pmes(&b)           print   "\n"  end  wrap   {   print   "Ho!   "   } |

Exceções   
•Objeto   
 da   
 classe   
 **Excep5on**,   
 ou   
 uma   
 subclasse •Interrompe   
 o   
 fluxo   
 normal   
 de   
 execução 

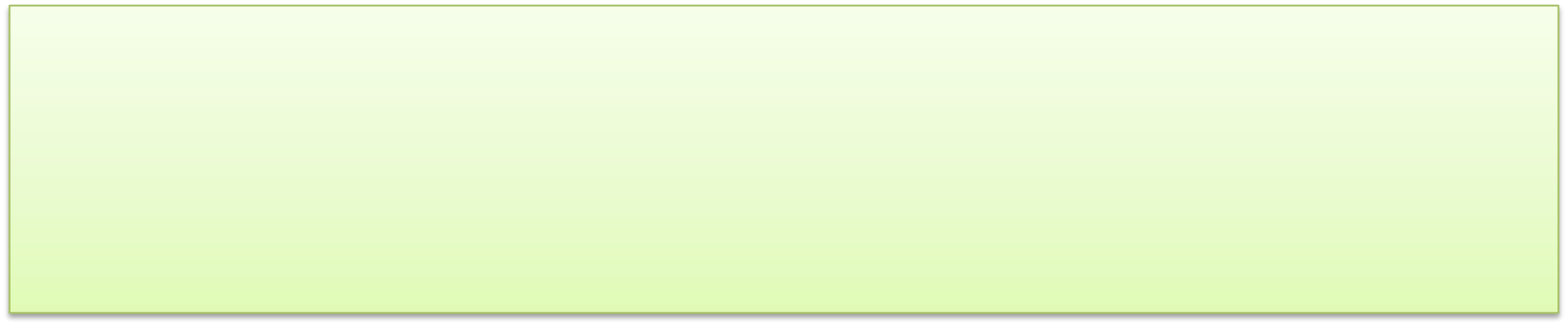
|  |
| --- |
| begin           content   =   load\_blog\_data(file\_name)  rescue   BlogDataNotFound           STDERR.puts   "File   #{file\_name}   not   found"  rescue   BlogDataFormatError           STDERR.puts   "Invalid   blog   data   in   #{file\_name}"  rescue   ExcepPon   =>   exc           STDERR.puts   "General   error   loading   #{file\_name}:   #{exc.message}" end |

Classes   
•Estrutura   
 similar   
 a   
 outras   
 linguagens   
 OO –Exemplo: 

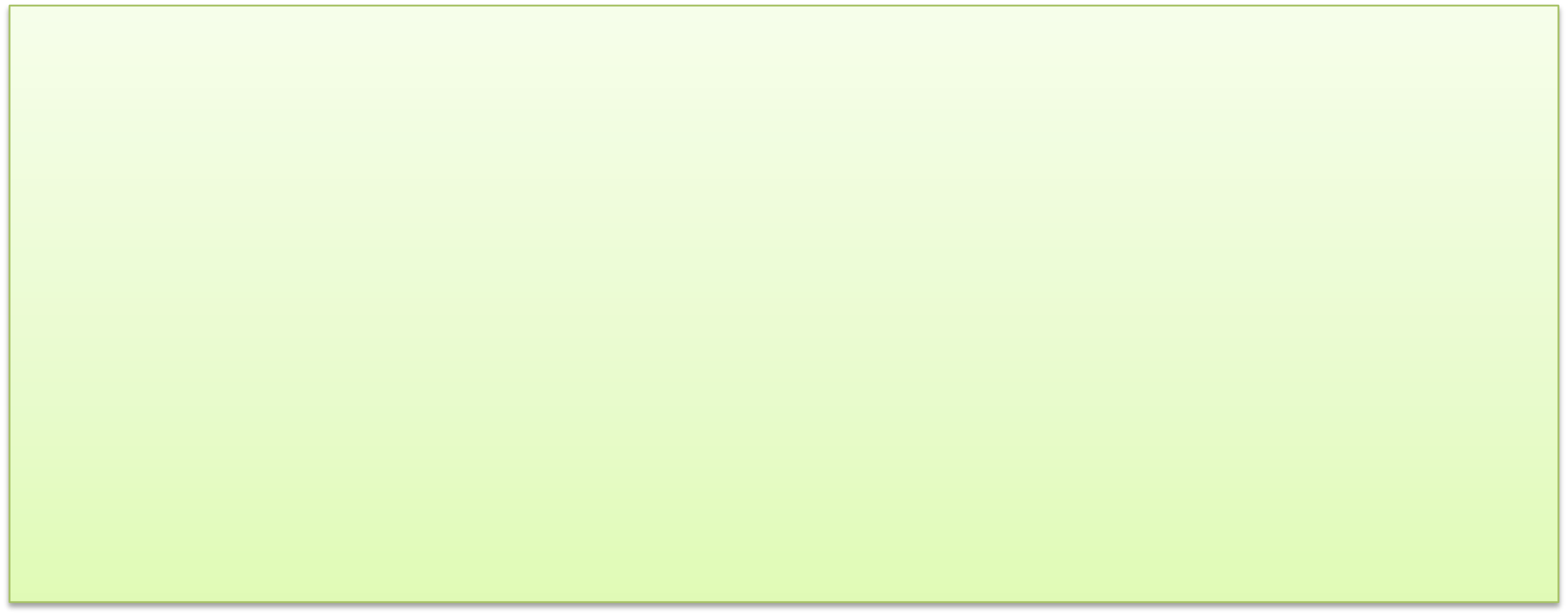
|  |
| --- |
| class   **Order**   <   AcPveRecord::Base           has\_many   :line\_items           def   self.find\_all\_unpaid                   self.where('paid   =   0')           end           def   total                   sum   =   0                   line\_items.each   {|li|   sum   +=   li.total}                 sum           end  end |

Classes   
•As   
 variáveis   
 de   
 instância   
 são   
 precedidas   
 de   
 @ 

|  |
| --- |
| class   Greeter           def   iniPalize(name)                   @name   =   name           end           def   name                   @name           end           def   name=(new\_name)                   @name   =   new\_name           end  end  g   =   Greeter.new("Barney") |

Classes   
•Métodos   
 acessadores   
 e   
 modificadores 

|  |
| --- |
| class   Greeter           a‚r\_accessor   :name                   *#   create   reader   and   writer   methods*          a‚r\_reader   :greePng               *#   create   reader   only*           a‚r\_writer   :age                                 *#   create   writer   only*  end |

Classes •Visibilidade   
 de   
 métodos 

|  |
| --- |
| class   MyClass           def   m1           end           protected           def   m2           end           private           def   m3           end  end |

Módulos

•Similares   
 a   
 classes,   
 visto   
 que   
 definem   
 uma   
 (i) coleção   
 de   
 métodos,   
 (ii)   
 constantes,   
 (iii)   
definição   
 de   
 classes   
 ou   
 (iv)outros   
 módulos

•Dois   
 propósitos   
 principais

–Atuam   
 como   
 “namespaces”

–ComparPlhar   
 funcionalidades   
 entre   
 classes

YAML   
•Acrônimo   
 de   
 *YAML   
 Ain’t   
 Markup   
 Language* •UPlizada   
 para   
 a   
 definição   
 de   
 configurações 

|  |
| --- |
| development:                   adapter:   sqlite3                   database:   db/development.sqlite3                   pool:   5                   Pmeout:   5000 |

Convertendo   
 Objetos   
•Ruby   
 permite   
 converter   
 um   
 objeto   
 em   
 um *stream*   
 de   
 bytes   
 –Visando   
 o   
 armazenamento   
 do   
 mesmo   
 –Processo   
 denominado   
 de   
 *marshaling*

•Rails   
 uPliza   
 esse   
 processo   
 para   
 armazenar   
 os dados   
 das   
 sessões   
 dos   
 usuários

DICA:   
**VALE   
 A   
 PENA   
 BUSCAR   
 MAIS**   
**INFORMAÇÕES   
 DA   
 LINGUAGEM   
 RUBY**