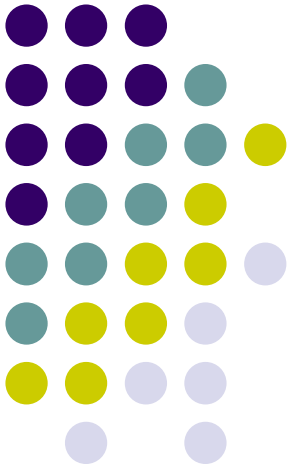
Язык Ruby

Денис С. Мигинский

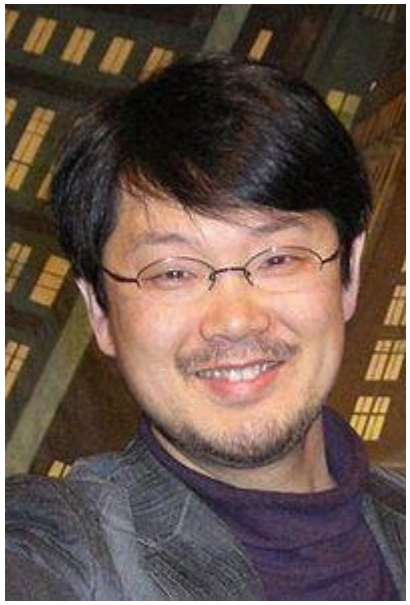


Ruby

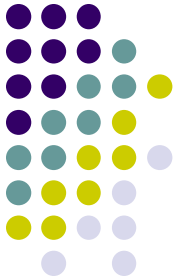
Создан Юкихиро Мацумото в 1995 г.

В основу положены элементы языков Perl, Python, Lisp, Smalltalk и др., а также «принцип наименьшего удивления».

Основные реализации:
http://www.ruby-lang.org
http://jruby.org

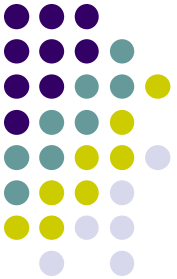


Основные характеристики Ruby



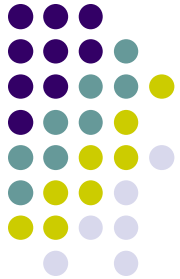
- Динамический язык
- Сквозная объектная модель
- Поддержка исключений
- Автоматическая сборка мусора
- Поддержка метапрограммирования (в т.ч. интроспекция, evaluate)
- Поддержка элементов функционального программирования (блоки/замыкания, λ-выражения)
- Встроенная поддержка регулярных выражений

Hello, world!



```
File.open("world_inhabitants.txt").
     readlines.
     each{|name|
          puts "Hello, #{name}!"
     }
```

Разделители выражений



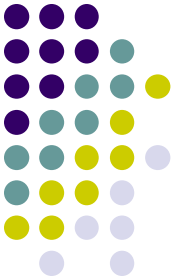
#комментарий

```
a=1; b=2    #конец выражения, «;» можно не ставить

c=a.to_f+ #выражение продолжается на следующей строке
      b.to_f

d=a.to_f\ #явный перенос на следующую строку
      -b.to_f
```

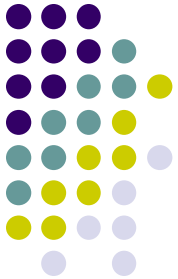
Определения и вызовы функций



```
def hello(name=“anonymous”)
     puts “Hello, #{name}!”
end

hello(“world”)
hello “Gustav”
hello
```

Строки (String)

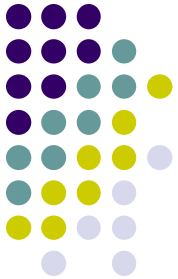


```
name="Richard I"

phrase="Hello, #{name}!"
phraseWrong='Hello, #{name}!'

puts phrase           #     Hello, Richard I!
puts phraseWrong      #     Hello, #{name}!
```

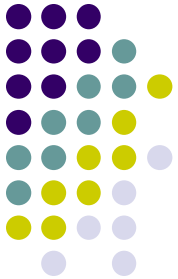
Интервалы (Range)



```
finish=3
(1..finish).each{|i| puts i}  #123

(1...finish).each{|i| puts i} #12
```

Массивы (Array)

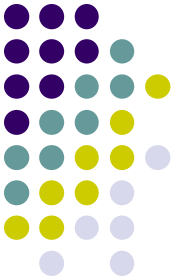


```
a = [10, 15, 20, 5]
puts a.max              #      20

b = [10, 30, “Richard I”]
puts b.max              #      Ошибка времени выполнения

c = Array.new(10){|i| i*i}
puts c                  #      0 1 4 9 16 25 36 49 64 81
```

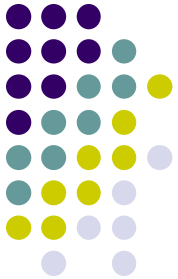
Ассоциативные массивы (Hash)



```
price={   “apple”=>10,
          “durian”=>100
}
price[“carrot”]=5

price.each{|fruit,price|
     puts "#{fruit} - #{price}“
}                    #       carrot – 5
                     #       durian – 100
                     #       apple – 10
```

Управляющие структуры: if, unless



#постфиксная форма

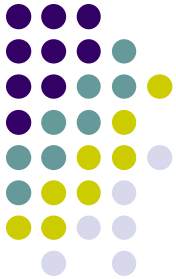
```
return 0 if a>0
return 0 unless a<=0
```

#"классическая" форма

```
if a>0
  return 0
else
  return 1
end
```

```
if a==0 then …
elsif a<0 then …
else …
end
```

Где искать истину?



Ложь:

- nil
- экземпляры FalseClass

Истина:

- все остальное

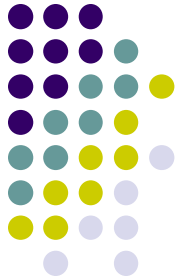
```
puts false.class          #FalseClass
puts true.class           #TrueClass

a=nil
puts a ? true : false     #false

a=false
puts a ? true : false     #false

a=0
puts a ? true : false     #true
```

Логические операторы



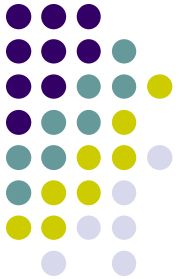
```
a=[1,2]
b=[3,4]

puts a||b                    #1 2
puts a&&b                    #3 4

puts a||nil                  #1 2
puts nil||b                  #3 4

puts a&&nil                  #nil
puts nil&&b                  #nil
```

Управляющие структуры: case

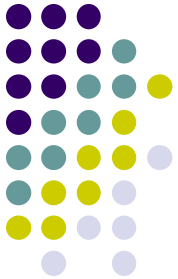


```
year = 1905
countryName = case year
  when 1721...1917 then “Russian Empire”
  when 1917...1991 then “USSR”
  else “Russian Federation”
end          #“Russian Empire”
```

#применяется оператор ===

```
puts (1721...1917) === 1905          #true
puts (1721...1917) == 1905           #false
```

Destructuring (реорганизующее присваивание)

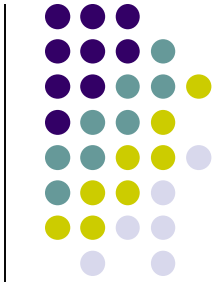


```
arr = [1, 2, 3, 4, 5]
a, b, _, *rest = arr
puts a, b, rest                    #1 2 4 5

def sum(first, *rest)
  if rest.empty?
    first
  else
    first + sum(*rest)
  end
end

puts sum *arr                      #15
```

Блоки и замыкания



Блок – часть кода с собственным контекстом (т.е. изолированными локальными переменными).

Блок как правило объявляется в одном контексте, а исполняется в другом.

Блок может взаимодействовать с тем контекстом, в котором объявлен, т.е. является замыканием.

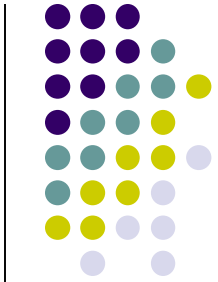
Аналогом блока в Java является анонимный класс.

Применение блоков: функционал

```
def nTimes (n)
  for i in (0...n)
    yield                    #вызов блока
  end
end

nTimes(5) {puts “Hello, world!”}
```

Применение блоков: оператор каррирования



```
def partial(firstVal, &fn)
  Proc.new {|*rest| fn.call(firstVal, *rest)}
  Proc.new {|*rest| yield(firstVal, *rest)}
end

puts partial(1) {|a,b| a+b}.call(2)
puts partial(1, &:+).call(2)
```

Определение и вызов блока

```
def func (&block)          #функция, вызывающая блок
  yield(<params>)          #вызов блока с параметрами
  block.call(<params>)     #--//--
end
```

#передача блока в качестве параметра функции

```
func(scalar_params){|<params>|<code>}
```

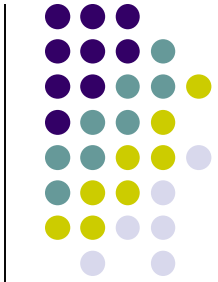
#определение блока, как объекта первого класса

```
block=Proc.new {|<params>| code}
block=lambda {|<params>| code}
```

#вызов самостоятельно определеннорго блока

```
block.call(<params>)
```

Контроль параметров вызова

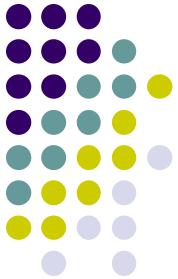


```
fn=Proc.new{|a,b|(a||0)+(b||0}
puts fn.call                    #0

fn=lambda{|a,b|(a||0)+(b||0}
puts fn.call                    #error

def fn(a,b)
  (a||0)+(b||0)
end
puts fn                         #error
```

Итераторы

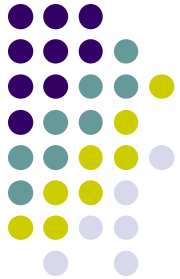


```
(2..5).reduce(1){|f, i| f*i}
```
#120 == 5!

```
(1..5).map {|i| i*i}
```
#[1, 4, 9, 16, 25]

```
File.open("world_inhabitants.txt").
     readlines.
     each{|name|
          puts "Hello, #{name}"
     }
```

module Enumerable: некоторые итераторы



```
all? [{|obj| block } ] => true|false

any? [{|obj| block } ] => true|false
```

#map == collect

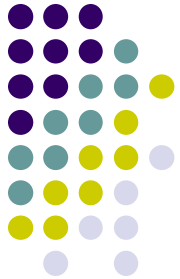
```
map {|obj| block } => array

select {| obj | block } => array
```

#reduce == inject

```
reduce(initial){|memo, obj| block } => obj
```

Задача 1



Задан набор символов и число **n**. Опишите функцию, которая возвращает список всех строк длины **n**, состоящих из этих символов и не содержащих двух одинаковых символов, идущих подряд.

Ограничения:

•решение должны быть выражено через map/reduce/select/reject и элементарные операции над строками и списками/массивами

Пример: для символов 'a', 'b', 'c' и n=2 результат должен быть ("ab" "ac" "ba" "bc" "ca" "cb") с точностью до перестановки.