Все является объектом, каждая операция - вызов метода.

Все является объектом, почти все - вызов метода.

•Даже малые целые числа и nil - объекты:

57.methods 57.heinz_varieties nil.respond_to?(:to_s)

• Перепишите каждый из этих методов для вызова send:

• Example: my_str.length => my_str.send(:length)

```
1 + 2 my_array[4]
my_array[3] = "foo" if (x == 3) ....
my_func(z)
```

```
1.send(:+, 2) my_array.send(:[], 4) my_array.send(:[]=, 3,"foo") if (x.
send(:==, 3)) ...
self.send(:my_func, z)
```

- Например, "неявное преобразование" при сравнении не является свойством языка, оно относится к instance методам.

Запомни!

- a.b означает: вызов метода b для объекта a
- a это получатель, которому ты отправляешь вызов метода, полагая, что a ответит на этот метод.
- *не означает:* b это переменная экземпляра a
- *не означает*: a это какая-то структура данных, в которой b является элементом

Понимание этого сильно облегчит вам жизнь

Пример: Каждая операция - вызов метода

```
y = [1,2]
y = y + ["foo",:bar] # => [1,2,"foo",:bar] y << 5 # => [1,2,"foo",:bar,5]
y << [6,7]           # => [1,2,"foo",:bar,5,[6,7]]
```

- “<<” *destructively изменяет* получателя, “+” - нет
- destructive методы часто оканчиваются на “!”
- Запомни, почти все эти методы относятся к массивам, а не к операторам языка.
- Так 5+3, "a"+"b", и [a,b]+[b,c] всё *разные* методы с именем '+'
- Numeric#+, String#+, и Array#+

Хэши и поэтический режим

```
h = {"stupid" => 1, :example=> "foo" } h.has_key?("stupid") # => true h ["not a key"] # => nil h.delete(:example) # => "foo"
```

- Идиома руби: “poetry mode”
- использует хэши в работе с “keyword-like” arguments
- не использует скобки, если последний аргумент- хэш
- не использует скобки вокруг функционвльных аргументов

```
link_to("Edit”,{:controller=>'students', :action=>'edit'}) link_to "Edit", : controller=>'students', :action=>'edit' link_to 'Edit', controller: 'students', action: 'edit'
```

- Если сомневаешься, ставь скобки

Поэтический режим в действии

a.should(be.send(:>=,7)) a.should(be() >= 7) a. should be >= 7

(redirect_to(login_page)) and return() unless logged_in?

redirect_to login_page and return unless logged_in?

```
def foo(arg,hash1,hash2)...
end
```

Which is *not* a legal call to foo():

```
Foo a, {:x=>1,:y=>2}, :z=>3
foo(a, :x=>1, :y=>2, :z=>3
foo(a, {:x=>1,:y=>2},{:z=>3}
foo a, {:x=>1,:y=>2},{:z=>3}
```

Классы и наследование

```
class SavingsAccount < Account   # inheritance
 # constructor used when SavingsAccount.new(...) called def initialize(starting_balance=0) # optional
 argument
   @balance = starting_balance
end
def balance                # instance method
   @balance  # instance var: visible only to this object
 end
 def balance=(new_amount) # note method name: like setter @balance = new_amount
 nd def deposit(amount) @balance +=amount
@@bank_name = "MyBank.com"   # class (static) variable
# class method
 def self.bank_name
# note difference in method def
   @@bank_name
end
# or: def SavingsAccount.bank_name ; @@bank_name ; end
```

Что является верным вызовом:

(a) my_account.@balance

(b) my_account.balance

(c) my_account.balance()

All three

Only (b)

(a) and (b)

(b) and (c)

Переменные: самый краткий путь

```
class SavingsAccount < Account def initialize
(starting_balance)
  @balance = starting_balance end
  def balance
    @balance
  end
  def balance=(new_amount) @balance
    =new_amount
end
```

```
class SavingsAccount < Account def initialize
(starting_balance)
 @balance = starting_balance end
 attr_accessor :balance

 end
```

```
class String def curvy?
   !("AEFHIKLMNTVWXYZ".include?(self.upcase)) end
end
```

```
String.curvy?("foo")
```

```
"foo".curvy?
```

```
self.curvy?("foo")
```

```
curvy?("foo")
```

ИТОГ: Ruby's Distinguishing Features (So Far)

- объектно-ориетирован с невозможностью мульти-наследования
- *все - объекты, даже целые числа и nil*
- классы и переменные невидимы вне класса
- все явлеяется вызовом метода
- обычно, важно только что получатель отвечает на вызов метода
- большинство операторов (like +, ==) являются instant methods
- Динамическое типирование : у объектов есть классы,у переменных нет
- Разрушающие методы
- Большинство методов неразрушающие
- Исключения: <<, some destructive methods (eg merge vs. merge! for hash)
- Idiomatically, {} and () sometimes optional

Международный банковский счет!

```
acct.deposit(100) # deposit $100 acct.deposit
(euros_to_dollars(20))
   # about $25
```

Международный банковский счет!

```
acct.deposit(100) # deposit $100 acct.deposit(20.
euros) # about $25
```

- нет проблем с открытием классов....

```
class Numeric
  def euros ; self * 1.292 ; end
end
```

Самоанализ и метапрограммирование

- Ты можешь задавать Руби объекты вопросы о них самих во время выполенния операции
- Ты можешь использовать эту информацию для нового кода
- Ты можешь снова открыть любой класс в любое время и добавить туда что-то
- Это *в дополнение* к extending/subclassing

Предположим, мы хотим перевести 5.euros.in(:rupees)

Какие изменения в Numeric будут самыми приемлемыми?

Change Numeric.method_missing to detect calls to 'in' with appropriate args

Define the method Numeric.in

Define the method Numeric.in

Change Numeric.method_missing to detect calls to 'in' with appropriate args

Циклы

```
["apple", "banana", "cherry"].each do |string| puts string
end
```

```
for i in (1..10) do puts i
end
```

```
1.upto 10 do |num| puts num
end
```

```
3.times {  print "Rah, " }
```

Если в итерации присутствует индекс, то ты что-то делаешь не так.

- *Итераторы позволяют объектам управлять обходом*

- (1..10).each do |x| ... end
(1..10).each { |x| ... }
1.upto(10) do |x| ... end => range traversal
- my_array.each do |elt| ... end => array traversal
- hsh.each_key do |key| ... end
hsh.each_pair do |key,val| ... end => hash traversal
- 10.times {...} # => *iterator of arity zero*
- 10.times do ... end

“Expression orientation”

```
x = ['apple','cherry','apple','banana']

x.sort # => ['apple','apple','banana','cherry'] x.uniq.reverse # => ['banana','cherry','apple'] x.reverse! # => modifies x

x.map do |fruit| fruit.reverse
#=> ['ananab','elppa','elppa','yrrehc'] x.collect { |f| f.include?('e') }
x.any? { |f| f.length > 5 }
```

- 10.times {...} # => *iterator of arity zero*
- 10.times do ... end

- настоящий пример

Какая строка не должна в итоге появиться в результате выполнения:

```
['banana','anana','naan'].map do |food|
  food.reverse
end.select { |f| f.match /^a/ }
```

naan

ananab

anana

The above code won’t run due to syntax error(s)