Въведение в Ruby

Най-добрият приятел на човека – irb

Итерактивна конзола, стартира се с `irb` от конзолата

```
$ irb
>> 6 * 9
54
>> "Ruby! " * 4
"Ruby! Ruby! Ruby! Ruby!"
>> -10.abs
10
```

Има и по-як еквивалент – `pry`

Извеждане на екрана

Става с `puts`:

```
puts "Chunky bacon!"
```

`puts` не е ключова дума, а "глобална" функция
Може да се извика и със скоби:

```
puts("Chunky bacon!")
```

Последното важи за всички функции в Ruby

Променливи

```
favourite = 'Chunky bacon' # локална

Tax = '20%'                # ”константа”

$important = 'Money'       # глобална
```

Java-style променливи

= не копира стойността, а насочва променливата към обекта отдясно

```
>> word = "oat"
>> breakfast = word
>> dinner = word
>> word.insert(0, 'g')
>> puts breakfast
goat
```

Ключови думи

```
alias     and       BEGIN     begin     break
case      class     def       defined?  do
else      elsif     END       end       ensure
false     for       if        in        module
next      nil       not       or        redo
rescue    retry     return    self      super
then      true      undef     unless    until
when      while     yield
```

(основни) Типове в Ruby (1)

Цели числа — `42`, `-271`, `1_024`

Низове — `"chunky"`, `'bacon'`

С плаваща запетая — `3.1415`

Булеви — `true` и `false`

Нищо — `nil`

Масиви — `[2, 3, 5, 7, 11]`

(основни) Типове в Ruby (2)

Хешове — `{'one' => 1, 'two' => 2}`

Регулярни изрази — `/(bb)|([^b]{2})/`

Области — `2..6`, `'a'...'z'`

Символи — `:name`, `:send`

Анонимни функции — `lambda { |x| x ** 2 }`, `-> x { x ** 2 }`

Обекти — `Person.new`

Цели числа

```
0

1729

-271

1_000_000

0377          # осмични

0b0100_0010 # двоични

0xDEADBEEF   # шестнадесетични
```

Числа с плаваща запетая

```
0.0

3.14

-273.15

1.22e28

1_000.001
```

Текст

```
'Chunky bacon, I say!'

"Who is John Galt?"

"We're the knights who day \"Ni!\""

'\'0 freddled gruntbugly\' he begun'

'This text contains\nno newlines'

"This text contains\none newline"
```

Операции с текст

```
>> "Hello, " + " " + "chunky!"
→  "Hello,  chunky!"
>> "Ha! " * 3
→  "Ha! Ha! Ha! "
>> "%s bacon" % 'Chunky'
→  "Chunky bacon"
>> text = 'Chanky'
>> text[2] = 'u'
>> text << ' bacon'
>> puts text
Chunky bacon
```

Операции с текст: Episode II

```
"bacon\n".chop                  # "bacon"
"bacon".chop                    # "baco"
"bacon\n".chomp                 # "bacon"
"bacon".chomp                   # "bacon"
"Goodbye".include? 'oo'         # true
"Danube".length                 # 6
"oat".insert 0, 'bl'            # "bloat"
"bloat".sub "bl", "m"           # "moat"
"bacon".upcase                  # "BACON"
"BaCoN".downcase                # "bacon"
```

Символи

Малко особен тип в Ruby

`:something`, `:other` и `:elsys`

Прилича много на низовете в Java — immutable и интерниран
Различни приложения – например като ключ в хешове

Интерполация

Работи при двойни кавички, не работи при единични

```
character = 'Arthur'
answer = 42
puts "#{character} had it: #{answer}"
```

Оператори

```
**   ~    /    %     +     -     &

<<   >>   |    ^     >     <     >=    <=    <=>

&&   ||   !    and   or    not   ||=   &&=

=    +=   -=   *=    /=    %=    **=   >>=   <<=

^=   &=   |=   ..    ...   defined?
```

Истината или се осмеляваш

- `nil` се оценява като "лъжа"
- `false` се оценява като "лъжа" (очевидно)
- всичко друго е истина
- `0`, `""` и `[]` са истина

Булеви оператори

`and`, `or` и `not` правят това, което очаквате

`&&`, `||` и `!` — също

Сравнения

```
>> 20 > 20
→  false
>> "perl" < "ruby"
→  true
>> 5 <= 5
→  true
>> "foo" <=> "bar"
→  1
```

Равенства

В Ruby има четири оператора за равенство

```
one == two
one === two
one.eql? two
one.equal? two
```

Равенства: == и equal?

Това е стандартното поведение на Ruby. Има специални случаи.

```
one = 'chunky bacon'
two = 'chunky bacon'
one == two              # true
one.equal? two          # false
```

Равенства и интерниране

```
one, two = 42, 42
one == two        # true
one.equal? two    # true
```

Същото важи и за символи.

Условия: ако и ако пък

```
if hungry?
  eat
else
  drink
end
```

```
if hungry?
  eat
elsif thirsty?
  drink
else
  philosophize
end
```

Условието като израз

Всяко нещо в Ruby е израз и се оценява на нещо:

```
classification = if age < 13
                   'young person'
                 elsif age < 20
                   'teenager'
                 else
                   'old dude'
                 end
```

Условия на един ред

```
eat if hungry?

order = if hungry? then 'food' else 'coffee' end
```

Условия: освен ако

```
unless tired?
  go_out_and_run
end
```

За `unless` важат всички неща за `if`

Списъци == масиви

Класът се казва `Array`. Разбира се, има литерален синтаксис

Дефинират се така: `[1, 2, 3, 4]`

Хетерогенни: `[18, :female, 'Burgas']`

Напълно mutable

Могат да се влагат, както всички колекции
Неограчиен, нестатичен размер

Пазят референции, а не копия към елементите си
Имплементирани са като C масив

Списъци: индексиране

```
numbers = [:zero, :one, :two]
numbers[1]   # :one
numbers[10]  # nil
numbers[-1]  # :two

numbers[5] = :five
numbers[5]   # :five
numbers      # [:zero, :one, :two, nil, nil, :five]
```

Списъци: `fetch`

`Array#fetch` хвърля грешка или връща друга стойност, при индексиране извън обема на списъка:

```
numbers = [:zero, :one, :two]

numbers.fetch(1)              # :one
numbers.fetch(10, :dunno)     # :dunno
numbers.fetch(10)             # error: IndexError
```

Списъкът и неговите методи

```
numbers = [3, 1, 2, 4]
numbers.length  # 4
numbers.size    # 4
numbers.sort    # [1, 2, 3, 4]
numbers.reverse # [4, 2, 1, 3]
numbers[1..2]   # [1, 2]
```

`sort` и `reverse` връщат нов списък, без да променят `numbers`.

```
prime_digits = [2, 3, 5, 7]

prime_digits.include? 2    # true
prime_digits.include? 4    # false
```

Операции със списъци

```
[:a, :b, :c] + [:d, :e]                  # [:a, :b, :c, :d,
  :e]
[:a, :b, :c, :b, :a] - [:b, :c, :d] # [:a, :a]
[:a, :b, :c] & [:b, :c, :d]              # [:b, :c]
[:a, :b, :c] | [:b, :c, :d]              # [:a, :b, :c, :d]
```

`&` и `|` конкатенират списъците и премахват повторенията.

Мутиране на списъци

```
numbers = [1, 2, 3]

numbers << 4
p numbers    # [1, 2, 3, 4]

numbers.insert 0, :zero
p numbers    # [:zero, 1, 2, 3, 4]

result = numbers.delete_at(0)
p result     # :zero
p numbers    # [1, 2, 3, 4]
```

Списъкът като стек

```
stack = [1, 2, 3]

stack.push 4
p stack          # [1, 2, 3, 4]

top = stack.pop
p stack          # [1, 2, 3]
p top            # 4
```

`#shift` и `#unshift` са аналогични, но работят с началото на списъка.

Още няколко метода

```
[1, 2, 3].join("-")          # "1-2-3"
[1, 2, 3].permutation        # сещате се какво връща
[1, 2].product([3, 4])       # [[1, 3], [1, 4], [2, 3], [2, 4]]
[[1, 2], [3, 4]].transpose # [[1, 3], [2, 4]]
[1, 2, 3, 4].shuffle         # разбърква списъка произволно
```

Последната запетая в списъка

```
songs = [
   'My Favorite Things',
   'Alabama',
   'A Love Supreme',
]
```

Понякога е доста удобно!

Списък от думи

Има специален синтаксис за списък от думи.

```
%w(chunky bacon)      == ['chunky', 'bacon']
%w[a b c]             == ['a', 'b', 'c']
%w{cool stuff}        == ['cool', 'stuff']
%w<coffee tea water> == ['coffee', 'tea', 'water']
%w|foo bar|           == ['foo', 'bar']
```

Може да използвате различни видове символи, които да ограждат списъка:

```
! @ # $ * - _
```

Този списък е непълен.

`Array#slice`

Всички списъци имат метод `slice`

"Операторът" за индексиране `[]` е всъщност метод на `Array`, който е синоним на `slice`

Може да приема индекс, област (range), както и индекс плюс брой елементи

Следователно може да връща един елемент или цял подсписък

Методът-setter `[]=` от своя страна позволява и да променяте въпросните отрязъци от списъка

Array#slice

```
numbers = [1, 2, 3, 4, 5, 6]

numbers[0..2]   # [1, 2, 3]
numbers[-3..-1] # [4, 5, 6]
numbers[1, 1]   # [2]

numbers[0..2] = [:wat]

numbers         # [:wat, 4, 5, 6]
```

Итериране на списъци

Итерира се с `#each`, както всичко останало в Ruby:

```
primes = [2, 3, 5, 7, 11]

primes.each { |n| puts n }

primes.each do |n|
  puts n
end
```

Забравете за оператора `for`. Използването му се приема за лош стил.

Итериране на списъци: `next` и `break`

`next` прескача изпълнението до края на блока и продължава със следващия елемент, ала `continue` в `C/Java`.

`break` прекъсва итерацията и продължава изпълнението след блока.

```
numbers.sort.each do |number|
  next if number.odd?
  break if number > 100

  puts number
end
```