

Программирование на Ruby
для ЛИНГВИСТОВ
a.k.a.
Ruby for Smart Linguists
Basic Ruby (w/o classes)

About

- Создатель: Yukihiro Matsumoto (matz)

- Первая версия в 1995

- Применяется в

консольные инструменты, *прототипирование*

GUI (есть библиотеки tk)

веб программирование Ruby on Rails



killer app

Версии Ruby

- ▶ прекращена поддержка версии 1.8.7
- ▶ текущие версии 1.9.x и 2.0.0

ruby-1.9.2-p290
ruby-1.9.3-p448
ruby-1.9.3-rc1
ruby-2.0.0-p247

- ▶ реализации Ruby

ruby :: MRI/YARV Ruby (The Gold Standard) {1.8.6,1.8.7,1.9.1,1.9.2...}
jruby :: JRuby, Ruby interpreter on the Java Virtual Machine.
rbx :: Rubinius
ree :: Ruby Enterprise Edition, MRI Ruby with several custom patches for performance, stability, and memory.
macruby :: MacRuby, insanely fast, can make real apps (Mac OS X Only).
maglev :: GemStone Ruby, awesome persistent ruby object store.

Осторожно, несовместимость!

- несовместимость между

ruby 1.8.7
ruby 1.9.x (1.9.2, 1.9.3)

good news!
ruby 1.9.x = 2.0.0

- ruby 1.8.7

```
"hello"[0] #=> 104
```

- ruby 1.9.x и 2.0.x

```
"hello"[0] #=> "h"
```

Совет!

решаясь перейти на
новую версию,
читать **ChangeLog**

Полезные утилиты - irb

- irb – интерактивный Ruby, консоль Ruby

```
> irb
```

REPL - read, evaluate, print, loop

- Запустите irb и попробуйте выполнить следующие команды

```
3+6      #=> 9
```

```
9/4       #=> 2
```

```
9.0/4     #=> 2.25
```

```
9.to_f/4  #=> 2.25
```

```
puts "hello " + "world"
```

```
num = 5  
puts num
```

эээ?

метод `to_f` преобразует во float
(число с плавающей точкой)

оператор `puts`
put string

{ набрать quit }

Задание

Задание: в irb, создайте переменную, содержащую строку "hello world"

```
> str = "hello world"
```

выделите из строки первую букву каждого из слов, объедините и распечатайте. Должно получиться "hw"

Решение:

```
str = "hello world"
```

```
puts str[0] + str[6]
```

Полезные утилиты - ri

- ri – (ruby information) консольная справочная система Ruby

```
> ri --help  
> ri --list  
> ri
```

- Выполните команды

```
> ri String  
> ri String#downcase
```

Метод downcase
объекта (экземпляра)
класса String

пример использования **метода объекта**: "Hello".downcase

```
> ri String.new  
> ri String.downcase
```

Метод new
класса String

пример использование **метода класса**: str = String.new

Задания

- Задание: запустите irb и в нем выполните преобразование строки к верхнему регистру. Какой **метод** надо применить к строке? Вставьте его вместо xxx:

```
puts "hello".xxx
```

```
ri String#upcase
```

- многоликий puts. этот метод определен во многих классах

```
> ri puts
```

- Задание: исследуйте отличия puts от print

```
puts "hello"; puts "world"
```

```
$, = ";
```

```
print "hello"; print "world"
```

```
print "hello", "world"
```


Переменные vs. Константы

➤ Переменные (изменяемые) vs константы (неизменяемые)

➤ Правила именования переменных и констант

- буквы [A-Za-z]
- цифры (не может быть первой) [0-9]
- нижнее подчеркивание _

```
name_1 = "Ruby"
```

➤ Переменная не должна начинаться с большой буквы.

➤ Вопрос: с чего может начинаться имя переменной?

Ответ: _[a-z]

Переменные vs. Константы

➤ Константы начинаются с большой буквы.

➤ попробуйте в irb

```
> puts RUBY_VERSION
```

`#=>1.9.2`

`lesson.02/test_stderr.rb`

```
STDOUT.puts  
STDERR.puts
```

➤ Задание: создайте свою константу

```
> ZZZ = 123  
> Qqq = 666
```

и присвойте им другое значение

```
> ZZZ = "reassigned"  
> Qqq += 2
```

oops!

```
(irb):6: warning: already initialized  
constant ZZZ  
=> "reassigned"
```

Константы

➤ Константами считаются имена классов и модулей

- String, Array, Hash
- Enumerable, Comparable
- MyOwnClass, Myownclass, My_own_class

➤ Убедитесь в этом, выполнив команду

```
> ri --classes
```

➤ Вопрос: Чем являются `_var` и `_Var`, переменными или константами?

```
_var = 9  
_Var = 99
```

пэрэмэнными

Типы данных

- ▶ Любой программный объект принадлежит к тому или иному типу

- ▶ Тип определяет

- ▶ допустимые значения и свойства
- ▶ перечень операций, применимых к значениям данного типа

- ▶ Некоторые типы данных

- ▶ численные: Integer, Float, Fixnum (<Integer), Bignum (<Integer), Numeric
- ▶ строковые String
- ▶ логические (булевские = boolean): FalseClass, TrueClass
- ▶ File, IO
- ▶ Symbol
- ▶ Array, Hash

Типы данных - 2

♦ Язык Ruby позволяет задавать свои собственные типы (определять классы) и снабжать их необходимыми свойствами.

- Dictionary
- PartOfSpeechDictionary
- Sentence
- Word
- AnnotatedWord

Типы данных - 3

- Языки делятся на языки с
 - динамической типизацией (shell, awk, ruby, python,...)
 - статической типизацией (C, Java)

- При статической типизации

- тип переменной задается сразу

`int a = 20`

- тип переменной нельзя изменить в процессе работы

`a = "now this is a string"`

- Ruby – язык с динамической типизацией

А в руби?

Аллилуйя!

Пример

- ▶ Выполните скрипт lesson.02/script_2.rb

```
#!/usr/bin/env ruby
```

```
a = 10  
puts a, a.class
```

```
a = "ten"  
puts a, a.class
```

```
a = "10"  
puts a, a.class
```

lesson.02/script_2.rb

метод class позволяет
узнать тип объекта

Задание

➤ Задание: выясните, к какому типу принадлежат следующие значения

3.14 3.14.class ==> Float

"3.14" "3.14".class ==> String, потому что в кавычках

[1, 2, 3] [1, 2, 3].class ==> Array, потому что в квадр. скобках

:Array :Array.class ==> Symbol, потому что начинается с :

1..5 (1..5).class ==> Range, потому что START..END

/[a-z]/ /[a-z]/.class ==> Regexp, потому что в слэшах

Задание

➤ Какой результат выполнения следующих операций?

$20 + 30$ $\#=> 50$

$"20" + "30"$ $\#=> "2030"$ конкатенация строк

$20 * 3$ $\#=> 60$ арифметическое умножение

$"20" * 3$ $\#=> "202020"$ String multiplication

$"20" * "3"$ $\#=> \text{TypeError: can't convert String into Integer}$

Присваивание (assignment)

- Оператор `=` служит для присвоения значения переменной

```
num = 42
```

- Параллельное присваивание

```
word, freq, tag = "apple", 42, "noun"
```

```
word, freq, tag = "apple", 42, "noun"
```

```
word = "apple"  
freq = 42  
tag = "noun"
```

- пример: `word, tag = "apple_NN".split(/_/)`

- Исследуйте, какие значения примут переменные:

```
a,b,c = 10,20
```

```
a,b,c = 10,20,30,40
```

Задание

- Задание: задайте две переменные (значение одной “Susan” а другой 25) и выведите текст

“her name is Susan and she is 25 years old”

Подсказка: в скрипте использовать оператор конкатенации строк +

```
#!/usr/bin/env ruby
```

```
name, age = “Susan”, 25
```

```
puts “her name is ” + name + “ and she is ” + age + “ years old”
```

```
puts “her name is #{name} and she is #{age} years old”
```



oops

`#{...}` интерполяция

Задание

- Решение 2: преобразование типов (привести к типу String)

```
#!/usr/bin/env ruby  
  
name, age = "Susan", 25  
  
puts "her name is " + name + " and she is " + age.to_s + " years old"
```

age.class

==> Fixnum

ri Fixnum#to_s

см. lesson.03/interpolation/test_puts.1.rb

Рюшечки: puts с шаблоном

- ▶ шаблон и позиционная подстановка

```
#!/usr/bin/env ruby
```

```
name, age = "Susan", 25
```

```
template = "her name is %s and she is %s years old"
```

```
puts template % [ name, age ]
```

см. также sudoku

```
> ruby sudoku_solver.rb
```

Оператор условия if

♦ if / then / end

```
if EXPRESSION [; then]
  # если EXPRESSION истинно,
  # то попадаем в эту ветку
  ...
end
```

; перед *then* необязательно

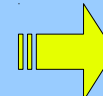
может употребляться в постпозиции:

```
puts "#{a} is a positive number" if a > 0
```

```
a = 5
```

```
if a > 0
```

```
  puts "#{a} is a positive number"
end
```



это true

```
if a > 0; then
```

```
  puts "#{a} is a positive number"
end
```

Оператор условия if

♦ if / then / end

```
if EXPRESSION [; then]
  # если EXPRESSION истинно,
  # то попадаем в эту ветку
  ...
end
```

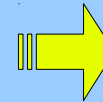
; перед *then* необязательно

может употребляться в постпозиции:

```
puts "#{a} is a positive number" if a > 0
```

a = 5

if a > 0



это true

```
  puts "#{a} is a positive number"
end
```

if a > 0; then

```
  puts "#{a} is a positive number"
end
```

if пошагово

```
a, b = 10, 10
```

```
if a > 0 && a == b  
  puts "hello"  
end
```

```
a, b = 10, 10
```

```
if true  
  puts "hello"  
end
```

```
a, b = 10, 10
```

```
if true && a == b  
  puts "hello"  
end
```

```
a, b = 10, 10
```

```
puts "hello"
```

проверка
на равенство
==

```
a, b = 10, 10
```

```
if true && true  
  puts "hello"  
end
```

"hello"

Оператор условия if

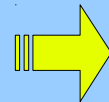
♦ if / then / else / end

```
if EXPRESSION [;then]  
  # сюда если TRUE  
else  
  # сюда если FALSE  
end
```

; перед *then* необязательно

```
a = - 5
```

```
if a > 0
```



это false

```
  puts "#{a} is a positive number"
```

```
else
```

```
  puts "#{a} is not a positive number"
```

```
end
```

Задание

➤ чему равны **a** и **b** после выполнения данного кода?

проверка
на НЕравенство
!=

```
a, b = 10, 10
```

```
if a > 0 && a != b
  b = b - 1
else
  b = b + 1
end
```

```
a, b = 10, 10
```

```
if true && a != b
  b = b - 1
else
  b = b + 1
end
```

```
a, b = 10, 10
```

```
if true && false
  b = b - 1
else
  b = b + 1
end
```

```
a, b = 10, 10
```

```
if false
  b = b - 1
else
  b = b + 1
end
```

```
a, b = 10, 10
```

```
if false
  b = b - 1
else
  b = b + 1
end
```

```
a, b = 10, 10
```

```
if false
  b = b - 1
else
  b = 11
end
```

Оператор условия if

♦ if / then / elsif+ / else / end

; перед *then* необязательно

```
if EXPRESSION1 [; then]
  ...
elsif EXPRESSION2 [; then]
  ...
elsif EXPRESSION3 [; then]
  ...
else
  ...
end
```

ветка else
необязательна

♦ если *EXPRESSION1* равен TRUE, то дальнейшие условия не проверяются

Сопоставление РВ

- Условные операторы, сопоставляющие (matching) строки:

оператор	описание	операнды	ыднарепо
<code>=~</code>	метчит	<code>string =~ regexp</code>	<code>regexp =~ string</code>
<code>!~</code>	не метчит	<code>string !~ regexp</code>	<code>regexp !~ string</code>

Например:

```
"paper" =~ /[a-z]/  
/[0-9]/  =~ "paper"
```

```
str = "paper"  
  
if str !~ /[a-z]/  
  puts "no letters"  
else  
  puts "contains letters"  
end
```

Сопоставление РВ

➤ Что возвращают следующие команды (в irb)?

"paper" =~ /[a-z]/ #=> 0

"Paper" =~ /[a-z]/ #=> 1

"Paper" =~ /[0-9]/ #=> nil

ri String# =~

возвращается позиция начала метча -
позиция первого символа,
который заметило регулярное выражение
(отсчет позиций с 0)

"paper" !~ /[0-9]/ #=> true

Что есть true и что есть false

- ♦ **Любое** выражение или объект в рубли имеет булевское (логическое) значение (true или false)

ПОЭТОМУ

любое выражение или объект можно использовать в условных конструкциях

```
if "Paper" =~ /[a-z]/  
  ...  
end
```



```
if 1  
  ...  
end
```



```
if true  
  ...  
end
```

- ♦ Вопрос: что произойдет в результате выполнения следующей команды?

```
if 1; then puts "is true"; else puts "is false"; end
```

Задания

► Выясните, какие из следующих значений считаются true в руби?

-5 true

0 true

в AWK false

“paper” true

“” true

в AWK false

true true

отсутствует в AWK

false false

отсутствует в AWK

nil false

отсутствует в AWK

Циклы

- Циклы позволяют выполнить какое-то действие/действия **несколько раз**.
- Сравните, сколько раз выполнится блок (lesson.03/while/test_while_1.rb)

```
a = 2
if a < 5
  b = a ** 2
  print b
end
```

`#=>4`

```
a = 2
while a < 5
  b = a ** 2
  print b, ", "
  a += 1
end
```

`#=> 4, 9, 16`

- Способы организации циклов в руби:
while (until), for, loop, times, upto, downto, step, each и его братья
- Другие операторы, управляющие циклами:
next, break; *retry*, *redo*

Цикл while

- Цикл while выполняется, пока условие истинно

```
while expression-that-is-TRUE [do]  
  # ....  
end
```

- Сколько раз выполниться следующий цикл (while/test_while_2.rb)?
Расскажите, как он выполняется:

```
i = j = 0  
  
while i < 5 && j < 5  
  puts "i=#{i}, j=#{j}"  
  i += 1  
  j += i  
end  
  
puts ""
```

i=0, j=0
i=1, j=1
i=2, j=3

Почему цикл не пошел на следующую итерацию?

i=3, j=6

Цикл while с IO#gets

- Вопрос: объясните, как работает следующий цикл?

например, обрабатывается файл, в котором одна строка "hello"

```
while line = gets      while line = "hello"      while "hello"      while true
  # действия          # ...
  # line.chomp!       end                        end
  ...
end
```

- Вопрос: В какой момент этот цикл остановится? читать: IO#gets

Ответ: из IO#gets: Returns +nil+ if called at end of file.

- Принудительное завершение цикла: break

```
while true; do
  ...
  break if some-condition
  ...
end
```

Чтение из потока ввода

```
ri IO#gets
```

- ▶ выполните скрипт `lesson.03/gets/test_gets.1.rb`
- ▶ Прочитайте скрипт и скажите, то ли выводится, что “хотел” сказать программист.

для сравнения, выполните `lesson.03/gets/test_gets.2.rb`

- ▶ Исправьте скрипт `test_gets.1.rb`, чтобы он работал аналогично `test_gets.2.rb`

```
ri String#chomp  
ri String#chop  
ri String#strip
```

```
ri String#chomp!  
ri String#chop!  
ri String#strip!
```

Задания

- Задание: напишите скрипт (test_numbers.rb), который просит пользователя ввести целое число и сообщает об этом числе, является ли оно положительным, отрицательным или нулем

```
#!/usr/bin/env ruby

msg = "Enter an integer number"
puts msg

while num = gets
  num = num.chomp.to_i

  # TODO: write your code here that tests the number
  if num ...

    puts msg
  end
end
```

test_numbers.rb

Ответ: lesson.03/test_numbers_done.rb

Задание

- Задание: Будет ли работать следующий скрипт?

```
#!/user/bin/ruby

while line = gets
  line.chomp!

  if line == "quit"
    exit
  elsif line < 0
    puts line + " is a negative number"
  else if line == "0"
    puts line + " is zero"
  elsif
    puts "#{line} is a negative number"
  end
end
```

Задание

➤ Задание: исправьте скрипт `lesson.03/test_numbers_buggy.rb`

в комментариях описано, что он должен делать

см. ответ в `lesson.03/test_numbers_correct.rb`

Задание

- на материале файла data/words.txt, подсчитайте скриптом, сколько
 - ★ в файле строк
 - ★ сколько слов, начинающихся с большой буквы
 - ★ сколько слов, начинающихся с маленькой буквы

ожидаемый выход как выход скрипта:

```
lesson.04/simple/count_words.rb
```

Совпадает ли количество слов 1. с суммой 2. и 3. ?

Если нет, то выведите строку/и, которая/ые не была/и подсчитана/ы?

Задание

- ♦ Из файла data/corpus.txt выведите непустые строки длиной меньше 10 токенов.
- ♦ Подсчитайте все непустые строки, пришедшие на вход, и все выведенные строки. Выведите эти счетчики в конце выполнения программы в поток ошибок

Ожидаемый выход как выход скрипта:

lesson.04/simple/short_paragraphs.rb

Начальный скрипт:

lesson.04/simple/short_paragraphs_stub.rb

```
while IO#gets
  String#length
  String#empty?
  String#split
  Array#length
```


Accuracy/Precision/Recall

		Gold	
		True (NP)	False (non-NP)
Auto	Pos. (NP)	tp: NP = NP	fp: NP != non-NP
	Neg. (non-NP)	fn: non-NP != NP	tn: non-NP = non-NP
		Precision	
		Recall	

tp - true positive
 fp - false positive
 fn - false negative
 tn - true negative

Accuracy:

$$(tp + tn) / (tp + tn + fp + fn)$$

Precision:

$$tp / (tp + fp)$$

Recall:

$$tp / (tp + fn)$$

Задание

♦ Задание: на основании файла `corpus_gold_vs_auto.txt` подсчитайте accuracy

ответ: `precision/compute_accuracy.rb`

Задание

- Задание: посчитайте точность распознавания NP.
- Дополнительно: округлите результат до двух знаков после запятой.

начальный скрипт: `precision/compute_precision_stub.rb`

ответ: `precision/compute_precision.rb`

Задание

- Задание: посчитайте посчитайте `recall` распознавания NP.
- Дополнительно: округлите результат до двух знаков после запятой.

ответ: `precision/compute_recall.rb`

- Задание: объедините в один скрипт вычисление всех метрик: `accuracy`, `precision`, `recall`

Задание

♦ Задание: измените скрипт `find_jj_with_jjr.rb` так, чтобы выход имел следующий вид:

NICE JJ --> JJR NICER

(т.е. пять полей разделенных табуляцией)

ответ: `lesson.06/find_jj_with_jjr.2.rb`

Ожидаемый выход в

`lesson.06/find_jj_with_jjr.2.out`

`lesson.07/find_jj_with_jjr.2.out`

Есть ли в выходе это две строки?

FAR JJ	-->	JJR FARTHER
FAR JJ	-->	JJR FURTHER

Задание

➤ Задание: (см. lesson.06/irrverbs/README) Разработайте скрипт, который находит в словаре просао глаголы, имеющие неправильную форму VBD, и выводит найденное в следующем формате:

GIVE	VB	-->	VBD	GAVE
SHED	VB	-->	VBD	SHED

(т.е. пять полей разделенных табуляцией)

Используйте DicTester как источник данных. см. пример выхода DicTester в файле:

lesson.06/irrverbs/dictester.txt

ответ: lesson.07/irrverbs/find_vb_irrvbd.rb

ожидаемый выход: lesson.07/irrverbs/find_vb_irrvbd.out

“Найди отличия”

1. Правильный ли это способ получить форму VB?

```
vb = line.split.first
```

см. файл dict.takeplace.txt

```
“TAKE PLACE classes: VB-134/10”.split  
=> ["TAKE", "PLACE", "classes:", "VB-134/10"]
```

2. Что изменится, если заменить регулярное выражение?

```
/^(.+[^\s])\s+classes:/'
```



```
/^(.+)\s+classes:/'
```

Работает ли скрипт?

Сравните значения переменной vb в обеих реализациях.

```
puts vb + ">"
```

“Найди отличия”

3. Что если изменить порядок проверок условий?

```
line =~ /VB-/ && line =~ /^(.+)\s+classes:/
```



```
line =~ /^(.+)\s+classes:/ && line =~ /VB-/
```

➡ Чему равно значение переменной `vb` в каждом из случаев?

4. Нужна ли проверка `&& vb` ? Сравните выход скрипта с и без этой проверки

```
if line =~ /VB-.+--> VBD-\d+ (.+)/ && vb
```

см. файл dict.do.txt

➡ paradigm of DO has no VB but has VBD

Массивы (Arrays)

- Arrays = массивы = списки
- Массивы один из базовых типов данных

```
letters = ["a", "b", "c", "d", "e"]
```

- Массив это структура данных, содержащая **ряд** объектов, доступ к которым определен **по индексу**.

- обычная переменная (не массив) имеет **одно** значение

```
age = 25
```

- В руби массивы индексируются начиная с 0.

ключ	значение
0	"a"
1	"b"
2	"c"
3	"d"
4	"e"

Массивы (Arrays)

- Обращение к элементу массива происходит через указание его индекса

```
letters = ["a", "b", "c", "d", "e"]
```

```
puts letters[0]      "a"
```

```
puts letters[3]      "d"
```

- В руби в массиве можно хранить данные разных типов

```
things = [1, "uno", 36.6, ["one", "two"], 1..10]
```

- Вопрос: что хранится в массиве things под индексом 3?

```
puts things[3]      ["one", "two"]      # массив строк
```

Обращение к элементу(ам) массива

- Дан массив

```
letters = ["a", "b", "c", "d", "e"]
```

- По индексу от начала массива

```
letters[1]      => "b"
```

```
letters[1] = 'B'
```

```
ri Array#[]  
ri Array#[]=
```

- По индексу считая с конца массива (нумерация начинается с -1)

```
letters[-1]      => "e"      # последний элемент массива
```

```
letters[-2]      => "d"      # предпоследний элемент массива
```

Задания

- Задание: как еще можно выделить первый/последний элементы массива?

```
ri Array
```

```
ri Array#first  
ri Array#last
```

```
letters.first  
letters.last
```

- letters.second, letters.third, ..., letters.onehundredtwentyfirst ?
- Задание: как выделить несколько элементов массива сразу? Как выделить элементы “b”, “c” и “d”?

```
letters = ["a", "b", "c", "d", "e"]
```

```
letters[1,3]          => ["b", "c", "d"]   # первый, длина
```

```
letters[2..4]         => ["c", "d", "e"]   # диапазон первый..последний
```

Как задать массив

- Перечислить через запятую значения его элементов

```
letters = [ "a", "b", "c", "d", "e" ]    digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Массив **строк** можно задать также таким образом

```
letters = %w{ a b c d e }
```

```
letters = %w( a b c d e )
```

- Преобразованием другого объекта в массив

строки:	"hello".split("")	#=> ["h", "e", "l", "l", "o"]
	"hello, world".split	#=> ["hello,", "world"]

диапазона:	(1..5).to_a	#=> [1, 2, 3, 4, 5]
	('a'..'e').to_a	#=> ["a", "b", "c", "d", "e"]

Добавление элементов в массив

- Объявляем массив

```
letters = Array.new
```

синоним: `letters = []`

- Здесь `new` это название метода класса, создающего новый экземпляр массива. Этот метод (*new*) называется *конструктором*.

- Добавление элемента в конец массива при помощи `<<`

```
letters << "a"
```

=> ["a"]

```
letters << "a" << "b" << "c"
```

=> ["a", "a", "b", "c"]

синоним – метод `Array#push`:

```
letters.push "k"
```

```
letters.push("k", "l", "m")
```

```
letters.push "k", "l", "m" # можно без скобок
```

Присвоение значения элементам массива

- Можно назначить значение произвольному элементу массиву

```
letters[10] = 'zzz'
```

```
Array#[] =
```

- Задание: какой вид будет иметь массив после выполнения следующих действий

```
things = ['a', 'b']  
things << 'k' << 'l'  
things[10] = 'zzz'
```

Ответ: ["a", "b", "k", "l", nil, nil, nil, nil, nil, nil, "zzz"]

- Задание: проверьте, что произойдет, если выполнить следующие действия

```
words[10] = "hello"
```



сначала массив
надо объявить:
words = []

Присвоение значения элементам массива

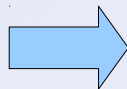
- Используя метод `[]` можно назначать значение сразу нескольким элементам массива (по аналогии с получением *нескольких* значений через метод `[]`)

```
letters = %w{a b c d e f g h}  
=> ["a", "b", "c", "d", "e", "f", "g", "h"]  
  
letters[1..3] = ["X", "Y", "Z"]  
letters  
=> ["a", "X", "Y", "Z", "e", "f", "g", "h"]
```

```
ri Array#[]=
```

- Что если длина диапазона (в индексе) не совпадает с длиной массива, который присваивается (справа от знака равно)?

```
letters[1..3] = [1,2]
```



```
=> ["a", 1, 2, "e", "f", "g", "h"]
```


Присвоение значения элементам массива

- Какой вид будет иметь массив после следующих действий?

```
numbers = (0...10).to_a
```

```
numbers[2..4] = [ [2, 'dos'], [3, 'tres'] ]
```

=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

=> [0, 1, [2, "dos"], [3, "tres"], 5, 6, 7, 8, 9]

Итераторы

- Чтобы пробежаться по всем элементам массива

```
letters = %w{ a b c d e f }
```

Array#each

```
letters.each do | val |  
  puts val  
end
```

ЭКВИВАЛЕНТНО

```
letters.each { |val|  
  puts val  
}
```

Array#each_index

```
letters.each_index do | idx |  
  puts "#{idx} = #{letters[idx]}"  
end
```

ЭКВИВАЛЕНТНО

```
letters.each_index { | idx |  
  puts "#{idx} = #{letters[idx]}"  
}
```

- Задание: попробуйте эти вкусные конструкции

Задания

- Задание: выведите элементы этого массива в обратном порядке

```
letters = %w{ a b c d e }
```

ri Array#reverse_each

```
letters = %w{ a b c d e }
```

```
letters.reverse_each do |item|  
  puts item  
end
```

ожидаемый
выход:

```
e  
d  
c  
b  
a
```

- Задание: и пронумеруйте элементы

bad news: no such thing as
Array#reverse_each_index

```
0 e  
1 d  
2 c  
3 b  
4 a
```

Задания

- Задание: как еще можно вывести массив в обратном порядке?

```
letters = %w{ a b c d e }
```

reverse ?

```
letters = %w{ a b c d e }
```

```
letters.reverse.each_with_index {|item, i|  
  puts "#{i} #{item}"  
}
```

```
0 e  
1 d  
2 c  
3 b  
4 a
```

ожидаемый
выход

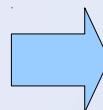
- У массива есть много методов, принимающих блок. Блок выполняется для каждого элемента массива. То есть, имеет место неявное итерирование по массиву.

Цикл for

- Цикл **for** синонимичен **Array#each**

```
letters = %w{a b c d e}
```

```
for i in letters  
  puts i  
end
```



```
a  
b  
c  
d  
e
```

- Страшная тайна: **for** вызывает метод **each**, поэтому **for** можно использовать с любым объектом, для которого определен метод **each**
- Что произойдет, если заменить массив на строку?

```
for i in "hello"  
  puts i  
end
```

```
NoMethodError: undefined method  
`each' for "hello":String
```

Задания

- Задание: Посчитайте длины всех слов в списке words.txt

примерный выход (см. lesson.08/word_lengths/word_lengths_1.out)

```
52 words of length 1  
183 words of length 2  
838 words of length 3  
3300 words of length 4
```

ответ:
lesson.08/word_lengths/word_lengths_1.rb

- Задание: то же самое, что и предыдущее задание, но выведите еще по 10 слов на каждую длину (10 первых встретившихся в списке слов)

примерный выход (см. lesson.08/word_lengths/word_lengths_2.out):

```
1 52, A, B, C, D, E, F, G, H, I, J  
2 183, Ac, Ag, Al, Am, Ar, As, At, Au, Av, Ba  
3 838, A's, AOL, Abe, Ada, Ala, Ali, Amy, Ana, Ann, Apr
```

ответ: lesson.08/word_lengths/word_lengths_2.rb

наш друг ri

➤ ri Array

= Class methods:

`[]`, `new`, `try_convert`

= Instance methods:

`&`, `*`, `+`, `-`, `<<`, `<=>`, `==`, `[]`, `[]=`, `abbrev`, `assoc`, `at`, `bsearch`, `clear`,
`collect`, `collect!`, `combination`, `compact`, `compact!`, `concat`, `count`, `cycle`,
`dclone`, `delete`, `delete_at`, `delete_if`, `drop`, `drop_while`, `each`, `each_index`,
`empty?`, `eql?`, `fetch`, `fill`, `find_index`, `first`, `flatten`, `flatten!`, `frozen?`,
`hash`, `include?`, `index`, `initialize_copy`, `insert`, `inspect`, `join`, `keep_if`,
`last`, `length`, `map`, `map!`, `pack`, `permutation`, `pop`, `pretty_print`,
`pretty_print_cycle`, `product`, `push`, `rassoc`, `reject`, `reject!`,
`repeated_combination`, `repeated_permutation`, `replace`, `reverse`, `reverse!`,
`reverse_each`, `rindex`, `rotate`, `rotate!`, `sample`, `select`, `select!`, `shelljoin`,
`shift`, `shuffle`, `shuffle!`, `size`, `slice`, `slice!`, `sort`, `sort!`, `sort_by!`, `take`,
`take_while`, `to_a`, `to_ary`, `to_s`, `transpose`, `uniq`, `uniq!`, `unshift`, `values_at`,
`zip`, `|`

Получение несоседних значений из массива

- Дан массив (см. lesson.09/data)

```
@months = %w[ Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec ]
```

- Загрузите этот массив из файла (выполнить в irb)

```
load 'data'
```

```
@months
```

```
ri Kernel#load
```

@ глобальная переменная

- Попробуйте загрузить таким же образом файл data.2 и доступиться к массиву days, который там определен.

- Задание: выделите одной командой все весенние и два первых осенних месяца, например, по их индексам в массиве

```
["Mar", "Apr", "May", "Sep", "Oct"]
```

```
ri Array#values_at
```

```
@months.values_at(2..4, 8, 9)
```

селекторами могут быть диапазоны или целые числа (положительные или отрицательные) ⁶⁴

Методы, оканчивающиеся на ? и !

- Имена методов могут *заканчиваться* на знаки ! и ?

Array#empty?
Array#include?

String#empty?
String#include?

- Вопрос: что это может обозначать? почитайте в ri описание разных методов с ?

описания в ri начинаются с **Returns true if ...**

- Такие методы (предикаты) всегда возвращают булевское **true** или **false**.

[]#empty?

#=> true, да массив пуст

%w[a b c].empty?

#=> false, нет, массив не пуст

arr = ["one", "two", "three"]

arr.member?('one')

#=> true

arr.include?(1)

#=> false

Синонимы
Array#member?
Array#include?

Опасные методы

- Знак ! (bang) обозначает, что метод “опасный”. Всегда есть метод без ! и метод с ! есть его “опасный” вариант
- Парные методы объектов класса Array

collect	collect!	map map!
---------	----------	----------

compact	compact!
---------	----------

flatten	flatten!
---------	----------

reject	reject!
--------	---------

reverse	reverse!
---------	----------

rotate	rotate!
--------	---------

select	select!
--------	---------

slice	slice!
-------	--------

sort	sort!
------	-------

sort_by	sort_by!
---------	----------

shuffle	shuffle!
---------	----------

uniq	uniq!
------	-------

Опасные методы - 2

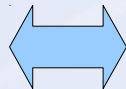
- В такой паре методов метод с ! изменяет *ресивер* (объект, у которого метод вызывается), а метод без ! возвращает другой объект, содержащий изменение:

```
arr = [1, 2, nil, 4]
```

- Сравните (в irb)

```
arr.compact  
#=> [1,2,4]
```

```
arr  
#=> [1, 2, nil, 4]
```



```
arr.compact!  
#=> [1,2,4]
```

```
arr  
#=> [1, 2, 4]
```

методы с ! тоже что-то возвращают

Опасные методы - 3

- ▶ Много других методов -- без ! -- также изменяют ресивер:

Array#delete

Array#push

пример Array#delete см. [lesson.09/find_vb_incomplete_pdg.2.rb](#)

- ▶ Очень распространенное неправильное обобщение: если метод изменяет ресивер, то он должен иметь ! .

Нет, это верно только для парных методов.

String#chomp

String#strip

String#chomp!

String#strip!

Задания

- Мысленно выполните скрипт `lesson.09/test_compact.rb`
- Вопрос: сколько элементов содержит массив `@months` в конце выполнения скрипта?

```
puts @months.length  
#=> 12
```

- Как удалить из массива все `nil`? Сколько элементов содержит сейчас массив `@months`?

```
@months.compact!  
puts @months.size
```

```
@months = @months.compact  
puts @months.length
```

```
Array#delete
```

```
Array#length  
Array#size  
Array#count
```

Строковое представление массива

- Преобразование массива в строку

Array#to_s

ruby1.8.7

```
puts [1,2,3,4].to_s  
#=> 1234  
  
puts [1,2,3,4].inspect  
#=> [1, 2, 3, 4]
```

ruby 1.9.x

```
puts [1, 2, 3, 4].to_s  
#=> [1, 2, 3, 4]  
  
puts [1,2,3,4].inspect  
#=> [1, 2, 3, 4]
```

Строковое представление массива - 2

➤ Преобразование массива в строку

```
Array#join(sep=$,)
```

\$, – output field separator
по умолчанию равно nil

```
arr = [1,2,3,4]
```

```
puts arr.join          #=> 1234
```

```
puts arr.join(', ')    #=> 1, 2, 3, 4
```

Строковое представление массива - 2

➤ Преобразование массива в строку

```
Array#join(sep=$,)
```

\$, – output field separator
по умолчанию равно nil

```
arr = [1,2,3,4]
```

```
puts arr.join          #=> 1234
```

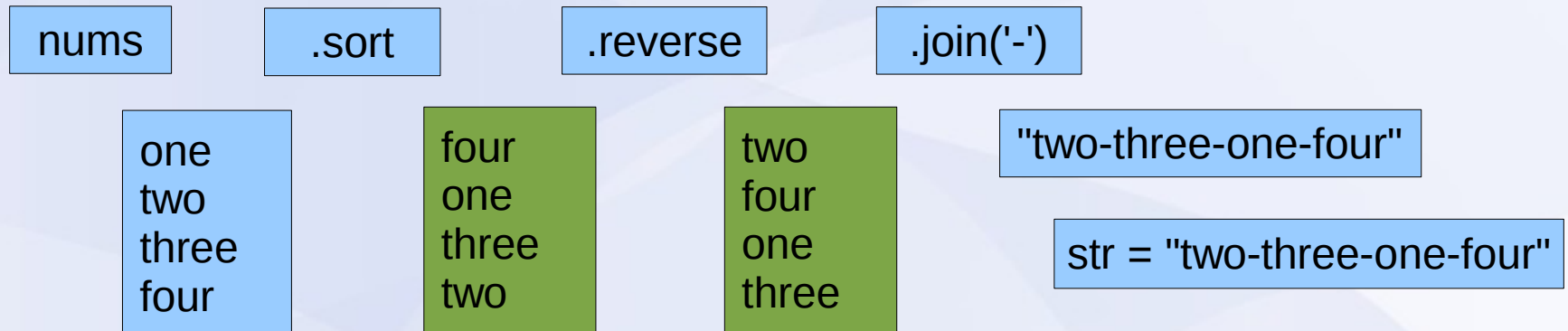
```
puts arr.join(', ')    #=> 1, 2, 3, 4
```


Method chaining

➤ Сцепление методов

```
nums = %w[one two three four]  
str = nums.sort.reverse.join('-')
```

str = "two-three-one-four"



- в процессе выполнения создаются промежуточные временные объекты
- Какой вид имеет массив `nums` после этих манипуляций?

массив `nums` не изменился

Задание

- Замените методы на их опасные варианты. Чему будет равно `str`? Какой вид будет иметь массив `nums` после этих манипуляций?

```
str = nums.sort!.reverse!.join('-')
```

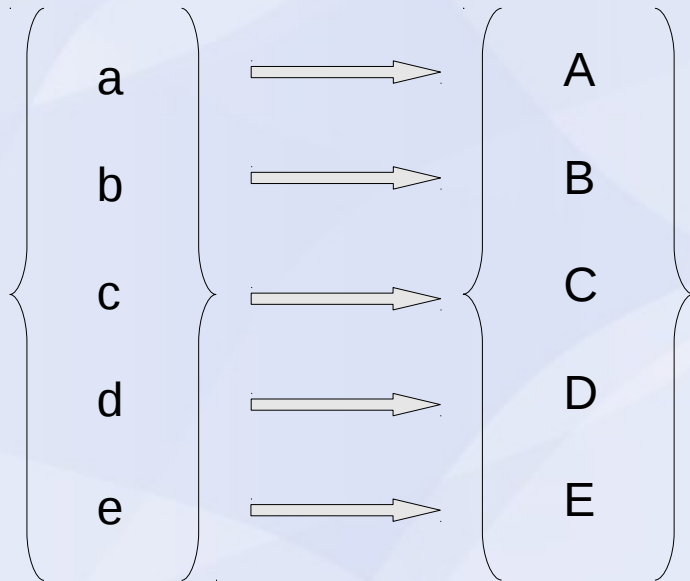
```
str = "two-three-one-four"
```

```
nums = ["two", "three", "one", "four"]
```

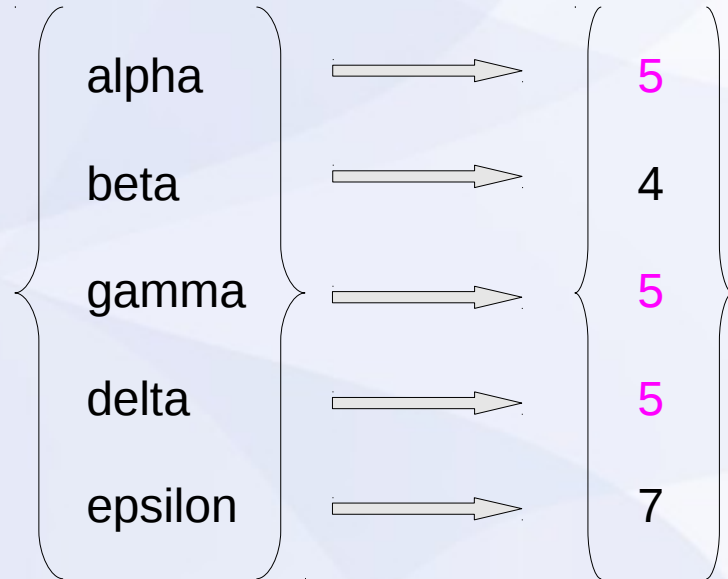
- Здесь `.sort!` и `.reverse!` не создают никаких временных объектов, а изменяют свой ресивер.

Отображение множества

- Отображение множества строчных букв на множество прописных



- Отображение множества слов на их длины



```
chars = %w{a b c d e}
upchars = chars.map do |char|
  char.upcase
end
```

Отображение множества

- Метод map (map!) обходит массив и для каждого элемента выполняет указанные действия, создавая из результата новый массив (или замещая текущий элемент этим результатом)

Синонимы

Array#map
Array#collect

Синонимы

Array#map!
Array#collect!

```
chars = %w{a b c d e}  
upchars = chars.map do |char|  
  char.upcase  
end
```

```
chars = %w{a b c d e}  
chars.map! do |char|  
  char.upcase  
end
```

- Что находится в массивах chars и upchars в обоих случаях?

chars – не изменился
upchars = ["A", "B", "C", "D", "E"]

chars = ["A", "B", "C", "D", "E"]
какой еще upchars? :-)

Array#map (cont'd)

- В блоке может быть больше одного действия.

см. lesson.10/test_map_1.rb

- Результат *последней* операции в блоке является результатом всего блока и именно это значение помещается в новый массив (или замещает прежнее значение, в случае Array#map!).

```
things = %w[ 1 uno 234 dos ]
things.map! do |el|
  if el =~ /\d+$/
    el.to_i
  else
    el.upcase
  end
  "HELLO"
end
```

см. lesson.10/test_map_things.rb

- Что произойдет в результате выполнения этого кода?

things = [1, "UNO", 234, "DOS"]

- Что произойдет, если добавить перед end "HELLO"?

["HELLO", "HELLO", "HELLO", "HELLO"]

Задания

- Измените скрипт `lesson.10/test_map_1.rb` таким образом, чтобы получить из массива `@numbers` двумерный массив вот такого вида:

`[["uno", "UNO"], ["dos", "DOS"], ["tres", "TRES"], ...]`

Ответ: `lesson.10/test_map_2.rb`

- Найдите максимальную и минимальную длину слов из `@numbers`

ожидаемый результат:

min: 3
max: 6

`Array#min`
`Array#max`

Ответ: `lesson.10/test_map_3.rb`

Состояние после маппирования:

`@numbers.map {|item| item.length}`
`#=> [3, 3, 4, 6, 5, 4, 5, 4, 5, 4]`

Задания

- Как иначе получить максимальное и минимальное значения (не используя методы `Array#min` и `Array#max`)?

```
@numbers.map do |item|  
  item.length  
end
```

`#=> [3, 3, 4, 6, 5, 4, 5, 4, 5, 4]`

```
@numbers.map do |item|  
  item.length  
end.sort
```

`#=> [3, 3, 4, 4, 4, 4, 5, 5, 5, 6]`

```
array.first  
array.last  
array[0]  
array[-1]
```

```
min, max = array.values_at(0, -1)
```

Задания

➤ Дано теггированное предложение. Необходимо вывести:

- предложение без тегов
- цепочку тегов (без слов)

Ограничение: нельзя использовать `String#gsub` на всем предложении, но можно использовать его для одного слова.

входной файл: `lesson.10/tagged.txt`

аккуратно преобразовать фразы:

`as_well_RB`

```
Array#split  
Array#index  
Array#rindex  
String#gsub
```

ответ: `lesson.10/untag.rb`

(Домашнее) Задание

- Даны теггированные отношения `r__VerbPhrase`. Необходимо вывести их без тегов.

Изучите структуру `r__VerbPhrase` (используйте метод `inspect`)

Возможно пригодятся:

```
Array#shift  
Array#unshift
```

входной файл: `lesson.10/tagged_relations.txt`

ответ: `lesson.10/untag_relations.rb`

Квантор всеобщности и квантор существования

- Квантор всеобщности – условие, верное для всех элементов множества

Array#all?

#=> true если **все** элементы удовлетворяют условию

#=> false если **хотя бы один** элемент **не** удовлетворяет условию

```
pets = %w{ bat dog cat cow wombat }
```

```
pets.all? do |pet|  
  pet =~ /a/  
end
```

- Все ли слова содержат букву а?

#=> false

- Каким будет результат следующей операции?

```
pets.all? do |pet|  
  pet =~ /[ieaou]/  
end
```

#=> true

Квантор существования

- ♦ Позволяет проверить, есть ли **хотя бы один** элемент, удовлетворяющий данному условию.

Array#any?

#=> true если **хотя бы один** элемент удовлетворяет усл.
#=> false если **ни один** элемент **не** удовлетворяет условию

- ♦ Когда надо выполнить действие, если среди тегов есть хотя бы один глагольный:

```
tags = [ "NN", "VBG", "JJing" ]  
  
if tags.any? { |tag| tag =~ /^V/ }  
  # do something  
end
```

Как не надо делать

- ▶ Когда надо выполнить действие, если среди тегов есть хотя бы один глагольный:

```
if tags.include?("VB") || tags.include?("VBZ")  
  || tags.include?("VBD") || tags.include?("VBN")  
  || tags.include?("VBG")  
then  
  # do something  
end
```

- ▶ Зачем здесь break?

Чтобы выполнить действие
только один раз

```
tags = [ "NN", "VBG", "JJing" ]  
tags.each do |tag|  
  if tag =~ /^V/  
    # do something  
    # ...  
    break  
  end  
end
```

Проверка на наличие элемента в массиве

```
pets = %w{ bat dog cat cow wombat }  
  
if pets.include?('dog')  
  ...  
end
```

Синонимы
Array#include?
Array#member?

➤ Объясните, почему это тоже работает аналогичным образом

```
if pets.index('dog')  
  # мы здесь  
end
```

```
if pets.index('aircraft')  
  # сюда мы не попадем  
end
```

Array#index(val)

Array#**r**index(val)

➤ возвращает позицию самой левой (первой) встречи val

➤ возвращает позицию самой правой (последней) встречи val

NB: позиция всегда отсчитывается от начала

Проверка на наличие подстроки в строке

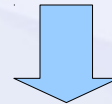
- Похожим образом работают одноименные методы в String

```
String#index  
String#rindex
```

- Закончите мысль

```
if "abrakadabra".index("k") ...  
  puts "В этом слове одна буква k"  
end
```

#=> 4



#=> 4

```
if "abrakadabra".index("k") == "abrakadabra".rindex("k")  
  puts "В этом слове одна буква k"  
end
```

- Как проверить, что в слове больше одной буквы b?

заменить == на !=

Задание

- Из файла `lesson.11/searching/text.txt` выведите предложения, содержащие хотя бы одно слово в середине, написанное в Titlecase.

Не используя `String# =~` на все предложение.
Представьте предложение в виде массива слов.

ответ: `lesson.11/finding/select_sent_with_titlecase_inside.rb`

ответ: `lesson.11/finding/select_sent_with_titlecase_inside.2.rb`

`Array#shift`

Selecting element(s)

- ♦ Найти и выбрать из массива элемент(ы), удовлетворяющие некоторому условию?
- ♦ Даны среднемесячные температуры в г. Надым. Как найти первую положительную температуру

temperatures = [-19.6, -18.2, -11.8, -7.6, 1.2, 10.6, 16.0, 12.1, 4.9, -19, -20, -25]

ответ в lesson.11/selecting/first_warm_month_temp.rb

```
temperatures.find {|el| el > 0}
```

`#=> 1.2`

Синонимы
Array#find
Array#detect

Selecting element(s) - 2

➤ Чем отличается Array#select от Array#find?

➤ Что вернет следующий код?

```
temperatures = [-19.6, -18.2, -11.8, -7.6, 1.2, 10.6, 16.0, 12.1, 4.9, -19, -20, -25]
```

```
temperatures.select {|temp| temp > 0}
```

```
#=> [1.2, 10.6, 16.0, 12.1, 4.9]
```

то есть, **массив** всех подходящих значений

```
temperatures = [-19.6, -18.2, -11.8, -7.6, 1.2, 10.6, 16.0, 12.1, 4.9, -19, -20, -25]
```

см. также

```
Array#grep
```

Сортировка массива, метод <=>

- Array#sort, Array#sort!

```
words = %w{ pear apple strawberry apple bears }
```

```
words.sort    #=> ["apple", "apple", "bears", "pear", "strawberry"]
```

- Элементы сравниваются между собой при помощи оператора <=> .

```
str <=> other_str
```

=> -1, 0, +1

“spaceship” operator

```
ri '<=>'  
ri 'String#<=>'
```

Возвращаемые -1, 0 или 1 показывают,
где по отношению к other_str сортируется str

- -1 если str сортируется **перед** other_str
- 1 если str сортируется **после** other_str
- 0 если равны

Оператор “космический челнок” <=>

- ♦ Что вернут следующие сравнения?

'apple' <=> 'apple' #=> 0

'apple' <=> 'bears' #=> -1 apple сортируется перед bears

'bears' <=> 'apple' #=> 1 bear сортируется после apple

'apples' <=> 'apple' #=> 1

- ♦ На метод <=> опираются все другие методы сравнения, определенные в модуле-примеси (mixin) Comparable:

< <= == >= >

Сортировка массива с блоком

➤ методы `sort` и `sort!` могут принимать блок, в котором описана процедура сравнения двух элементов. **блок должен возвращать -1, 0, 1**

➤ `Array#sort` (без блока) эквивалентен следующей команде с блоком

```
arr.sort do |a, b|  
  a <=> b  
end
```

два элемента попадают в переменные `a` и `b`
как взаимно расположить `a` и `b`?
если -1 или 0, то `ab`; если 1, то `ba`

➤ Как отсортировать массив в обратном порядке (от большего к меньшему), не используя `Array#reverse` ?

```
arr.sort do |a, b|  
  b <=> a  
end
```

```
sorted = arr.sort.reverse
```

см. `lesson.11/sorting/test_sort_1.rb`

Задания

- Отсортируйте массив @numbers по длине слов

```
@numbers.sort do |a, b|  
  a.length <=> b.length  
end
```

см. lesson.11/sorting/test_sort_2.rb

- Как этот же массив отсортировать в обратном порядке, от больших длин к меньшим?

```
@numbers.sort do |b, a|  
  a.length <=> b.length  
end
```

см. порядок аргументов в |b, a|

Сортировка по вычисленному значению

- `sort_by`, `sort_by!` производят сортировку по вычисленному значению

```
%w{ three one 1984 }.sort_by {|item|  
  item.length  
}
```

см. `lesson.11/sorting/test_sort_by_1.rb`

```
#=> ["one", "1984", "three"]
```

- Исследуйте, что делает скрипт `lesson.11/sorting/test_sort_by_2.rb`

Ответ: сортирует по согласным буквам

```
["ocho", "cinco", "cuatro", "dos", "diez", "uno", "nueve", "seis", "siete", "tres"]
```

Задания

- Измените test_sort_by_2.rb так, чтобы слова были отсортированы по количеству **гласных** в слове

["dos", "tres", "seis", "diez", "cinco", "uno", "ocho", "siete", "nueve", "cuatro"]

ответ см: lesson.11/sorting/test_sort_by_3.rb

- Отсортируйте массив @trn_numbers по немецким словам

```
@trn_numbers = [  
  [1, "one", "ein"],  
  [2, "two", "zwei"],  
  [3, "three", "drei"],  
  [4, "four", "vier"],  
  [5, "five", "fünf"]  
]
```

см. lesson.11/sorting/test_sort_3.rb
см. lesson.11/sorting/test_sort_by_5.rb

Пользовательские методы

Пользовательские методы

- Программист может задавать свои собственные методы
- Метод это способ сгруппировать код в одном месте
 - возможность абстрагировать от деталей реализации
 - возможность повторного использования (reusability)
 - более читабельный код
 - легче поддерживать
- синонимы: подпрограммы, функции, процедуры

Определение метода и его использование

➤ Метод должен быть *определен* до его *использования*

➤ Определение метода

```
def method_name(arg1, arg2....)
  # команды
  return ...
end
```

```
def method_name arg1, arg2....
  # команды
  return ...
end
```

➤ Использование (вызов) метода (method call):

```
res = method_name(a1, a2)
```

```
res = method_name a1, a2
```

➤ см. пример использования методов в

lesson.12/methods/extract_random_subcorpus.3.rb
lesson.12/methods/extract_random_subcorpus.4.rb

Аргументы методов

- Имена аргументов это *локальные названия* для внешних (по отношению к методу) переменных и литералов.

```
def unvowel(word)  
    word.delete('ieaou')  
end
```

```
unvowel("hello")
```

```
w = "good bye"  
unvowel(w)
```

внутри метода **unvowel** переменная **word** принимает значение "hello"

внутри метода **unvowel** переменная **word** принимает значение "good bye"

Аргументы методов

- Аргументы передаются позиционно

```
def max_of_three(a, b, c)
  if a > b && a > c
    return a
  elsif a > b && a < c
    return c
  elsif ...
    ...
  end
end

x, y = 1, 20
max_of_three(x, y, 10)
```

при вызове метода происходит

max_of_three(x, y, 10)



max_of_three(1, 20, 10)



def max_of_three(a, b, c)



max_of_three(a=1, b=20, c=10)

Задание

- ♦ Реализуйте метод `max_of_three` иначе.

ОТВЕТ:

```
lesson.12/methods/max_of_three.2.rb  
lesson.12/methods/max_of_three.3.rb
```

example of in-place unit testing:

```
puts max_of_three(1, 100, 2) == 100  
puts max_of_three(1, 100, 2, 500_000) == 500000
```

```
#=> true  
#=> true
```

Передача объектов в метод

- ▶ Объекты, передаваемые в метод через аргументы, передаются
 - ▶ по значению (по копии)
 - ▶ по ссылке
- ▶ Задание: исследуйте скрипт `lesson.12/methods/test_args_2.rb`.
Что произошло со строкой `str` и почему?
 - ▶ Метод `object_id` применяется к любой сущности в рубли, возвращая идентификатор этой сущности (объекта) в памяти.
- ▶ Задание: Исследуйте скрипт `lesson.12/methods/test_args_3.rb`.
Изменилось ли значение переменной `i` после вызова метода?
Как можно объяснить, что внутри метода переменная `i` сначала имеет один `object_id` а потом другой?

Это надо знать!

- Простые объекты (числа, true, false, nil) передаются по копии (в руби – при попытке их изменить, делается и изменяется копия).
 - Сложные объекты (String, Array, Hash, etc) передаются по ссылке -- такой объект можно изменить (в том числе по неосторожности).
- Это же отличие можно наблюдать в множественном присваивании:

два разных объекта a и b

```
a = b = 10
```

```
a += 10
```

```
b
```

```
#=> 20
```

```
#=> 10
```

две переменные aa и bb
ссылются на один и тот же
объект

```
aa = bb = [1,2,3]
```

```
aa << 4
```

```
bb
```

```
#=> [1,2,3,4]
```

```
#=> [1,2,3,4]
```

Область видимости переменных

- Область видимости (visibility scope) – фрагмент(ы) программы, где переменная видна (и ее можно использовать)
- Виды переменных
 - ➔ глобальные (\$zzz) – доступны везде: \$stdout, \$stderr, \$1..
 - ➔ локальные (без @ в начале)
 - ➔ **переменные объекта класса** (начинаются с @zzz)
 - ➔ переменные класса (начинаются с @@zzz)
- Метод создает свой собственный *локальный* контекст, переменные внутри метода никак не конфликтуют с переменными вне этого контекста, даже если их имена совпадают.
- Переменные с @ являются “глобальными” для скрипта и видны внутри всех методов, определенных в скрипте

Локальные переменные

- Локальная переменная видна только в локальном контексте

```
def increment(b)  
  b += 1  
end
```

} эта переменная b является локальной
для метода increment

```
b = 10
```

} эта переменная b является локальной
для скрипта (вне методов)

```
puts increment(b)  
puts b
```

`#=> 11`

`#=> 10`

- Это разные переменные b, они существуют в разных областях видимости

Переменные с @

- ▶ Переменная объекта класса (@name) видна во всем скрипте

```
def increment  
  @b += 1  
end
```

```
@b = 10
```

```
puts increment  
puts @b
```

```
#=> 11
```

```
#=> 11
```

- ▶ Задание: в скрипте `lesson.12/methods/extract_random_subcorpus.3.rb` сделайте переменную `pct` видимой внутри метода.

ответ: `lesson.12/methods/extract_random_subcorpus.5.rb`

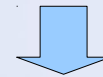
- ◆ Недостаточно заменить `pct` на `@pct`. Когда переменная стала глобальной для скрипта, нет необходимости передавать ее в метод как аргумент.
- ◆ Метод стал менее универсальным.

Return

- Метод может возвращать какое-либо значение. Для этого используется ключевое слово **return**
 - ➔ возвращает указанное значение
 - ➔ и выходит из метода
- В руби при помощи return можно вернуть любое количество любых объектов (руби объединяет их в массив)

```
def useless_method  
  a = 111  
  b = 222  
  return a, b, 42  
end
```

```
x, y, z = useless_method
```



```
x, y, z = 111, 222, 42
```

~~Come back~~ Return

- Что делает этот метод?

```
def longer_word(word1, word2)

  if word1.length > word2.length
    return word1
  end

  return word2

  puts "hello" # this never happens
end
```

- return выходит из метода

- Что будет напечатано?

```
w = "books"
puts longer_word("book", w)

#=> books
```

- Чему равно res ?

```
w = "burn"
res = longer_word "book", w

res = "burn"
```

Задания

- Разработайте метод, который принимает массив чисел и возвращает минимальное и максимальное значения.

```
values = [3,2,5,9,1,-7]  
mn, mx = min_max(values)
```

выход:
min = -7
max = 9

ответ `lesson.12/methods/min_max.1.rb`

- Объясните, что вы видите в `lesson.12/methods/min_max.2.rb`?

Задание

- Определите метод `select_by_length`, который из заданного массива выбирает слова заданной длины

```
dict = %w{cat act book teacher Ruby}  
res = select_by_length(dict, 4)
```

Ожидаемый результат:

```
#=> ["book", "Ruby"]
```

ответ: lesson.12/methods/select_by_length.rb

Методы без Return

- Метод не обязательно должен что-либо возвращать
 - ➔ метод изменяет сам объект, переданный ему как аргумент
 - ➔ метод выполняет какое-то действие, возвращаемое значение которого *не важно*

```
def greet(name)
  puts "Hello, #{name}."
end
```

`greet "Zeus"`

`greet('Apollo')`

- В руби метод без явного *return* возвращает результат последней операции!
- Вопрос: будет ли напечатан вопрос про гору Олимп?

```
if greet("Zeus")
  puts "How is the life on the Mount Olympus?"
end
```

не будет, так как
puts возвращает nil,
а nil это false

Параметры по умолчанию

- Аргументам метода можно задавать значение по умолчанию

```
Array#join(sep=$,)
```

```
arr = %w{uno dos tres}
puts arr.join      #=> unodostres
puts arr.join(', ') #=> uno, dos, tres
```

- Допускается любое количество опциональных аргументов при условии, что они являются *последними* аргументами в методе

```
def strjoin(a, b, c=nil, s=" ")
  [a,b,c].compact.join(s)
end
```

```
puts strjoin("aa", "bb", "cc", "\t")  #=> aa  bb  cc
puts strjoin("aa", "bb", "cc") + "!"  #=> aa bb cc!
puts strjoin("aa", "bb") + "!"        #=> aa bb!
puts strjoin("aa", "bb", "\t") + "!"  #=> aa bb \t!
```

- Вопрос: Что напечатают следующие команды?
- Вопрос: как напечатать “aa bb” разделенные табуляцией?

```
puts strjoin("aa", "bb", nil, "\t")  #=> "aa  bb"
```


Переменное количество аргументов

- В том случае, если функция должна иметь возможность вызываться с разным количеством элементов, используется * (splat operator)

```
def method_name( *args )  
  # args is an Array  
  # args[0], args[1] ...  
end
```

Использование:

```
method_name(1)  
method_name(1, "aa")  
method_name(1, "aa", x, y)
```

- Задание: разработайте метод *strjoin*, который производит конкатенацию заданных строк в одну через заданный сепаратор, принимая любое число строк для конкатенации в качестве аргументов.

```
puts strjoin("aa", "bb", ",")      #=>aa,bb  
puts strjoin("aa", "bb", "cc", "\t") #=>aa  bb  cc
```

ответ: lesson.13/strjoin.rb

Задание

- Будет ли работать такой код?

```
def strjoin(*args, sep="\t")  
  args.join(sep)  
end
```

см. lesson.13/strjoin.rb

Какой будет результат в случае:

```
strjoin("aa", "bb", "cc", "\t")  
strjoin("aa", "bb", "cc", "dd")
```

“\t” и “dd” относятся
к args или к sep?

(и снова) Циклы

- Ранее изученные циклы и методы для итерирования:

while for ... in ... Array#each Array#reverse_each

- По диапазону

```
(3..7).each {|i| puts i}
```

#=>

3
4
5
6
7

- Как вывести числа из диапазона в обратном порядке?

см. lesson.13/loops/test_range_each.rb

Методы upto/downto

- Цикл от одного заданного значения до другого заданного с шагом 1

```
3.upto(7) do |n|  
  puts n  
end
```

#=>

3
4
5
6
7

```
Integer#upto  
String#upto  
Date#upto
```

- downto – от большего к меньшему

```
7.upto(3) do |n|  
  puts n  
end
```

#=>

7
6
5
4
3

см. lesson.13/loops/test_upto.1.rb

пример с Date#upto
lesson.13/loops/test_upto_date.rb

- Исследуйте скрипт lesson.13/loops/test_downto.2.rb. Нужно ли брать в скобки (если да, то зачем):

```
(words.length-3).downto { |i| ... }
```

Метод (который танцует) step

- Что делает метод step?

см. lesson.13/loops/test_step_1.rb

```
1.step(20, 3) do |n|  
  print n.to_s + '  
end
```

#=> 1 4 7 10 13 16 19

перебор значений с шагом 3

- Задание: измените test_step_1.rb так, чтобы было выведена следующая последовательность (в обратном порядке):

20 17 14 11 8 5 2

```
1.step(20, -3) do |n|  
  print n.to_s + '  
end
```

ответ в lesson.13/loops/test_step_2.rb:

Метод step

- Метод step определен для классов Range, Numeric, Date

ri step

ri Numeric

- Почему метод upto определен для (под)класса Integer, а метод step для родительского (супер)класса Numeric?

3.14.upto(9.8) { |n| block }

Integer < Numeric

Numeric < Object

Неясно, какое должно быть следующее число за 3.14 – 3.141 или 3.15

Задание

- Реализуйте метод сортирующий массив чисел по алгоритму сортировки вставкой. Метод должен принимать на вход один аргумент - массив чисел – и возвращать *новый массив*, в котором эти числа отсортированы в восходящем порядке.

insert_sort [9,7,9,1,5,-3] #=> [-3,1,5,7,9,9]

Array#insert

начальный скрипт: lesson.13/insert_sort/insert_sort_to_new_stub.rb

ответ: lesson.13/insert_sort/insert_sort_to_new.rb

взять очередной элемент из массива	положить в подходящее место в новом массиве
[9, 7, 9, 1, 5, -3]	[9]
[9, 7, 9, 1, 5, -3]	[7, 9]
[9, 7, 9, 1, 5, -3]	[7, 9, 9]
[9, 7, 9, 1, 5, -3]	[1, 7, 9, 9]
[9, 7, 9, 1, 5, -3]	[1, 5, 7, 9, 9]
[9, 7, 9, 1, 5, -3]	[-3, 1, 5, 7, 9, 9]

Задание

- ♦ Реализуйте метод, производящий сортировку массива чисел на месте (in place - переупорядочивается сам массив непосредственно). Используйте алгоритм сортировки вставкой.

Алгоритм и псевдокод:

http://ru.wikipedia.org/wiki/Сортировка_вставкой

начальный скрипт: `lesson.13/insert_sort/insert_sort_in_place_stub.rb`

ответ: `lesson.13/insert_sort/insert_sort_in_place.rb`

Заголовок