

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"
```



`#{...}` 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

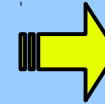
может употребляться в постпозиции:

```
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
```

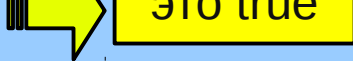
; перед *then* необязательно

может употребляться в постпозиции:

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

a = 5

```
if a > 0
  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

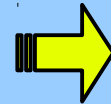
♦ if / then / else / end

```
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

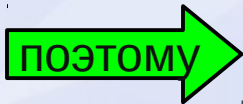
ri String# =~

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)



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                   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

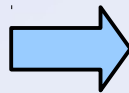
- Используя метод `[]=` можно назначать значение сразу нескольким элементам массива (по аналогии с получением *нескольких* значений через метод `[]` )

```
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
```

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

- Задание: и пронумеруйте элементы

bad news: no such thing as  
Array#reverse\_each\_index

# 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

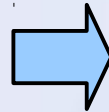
- У массива есть много методов, принимающих блок. Блок выполняется для каждого элемента массива. То есть, имеет место неявное итерирование по массиву.

# 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)



# 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

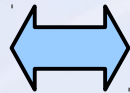
- В такой паре методов метод с ! изменяет *ресивер* (объект, у которого метод вызывается), а метод без ! возвращает другой объект, содержащий изменение:

```
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
```

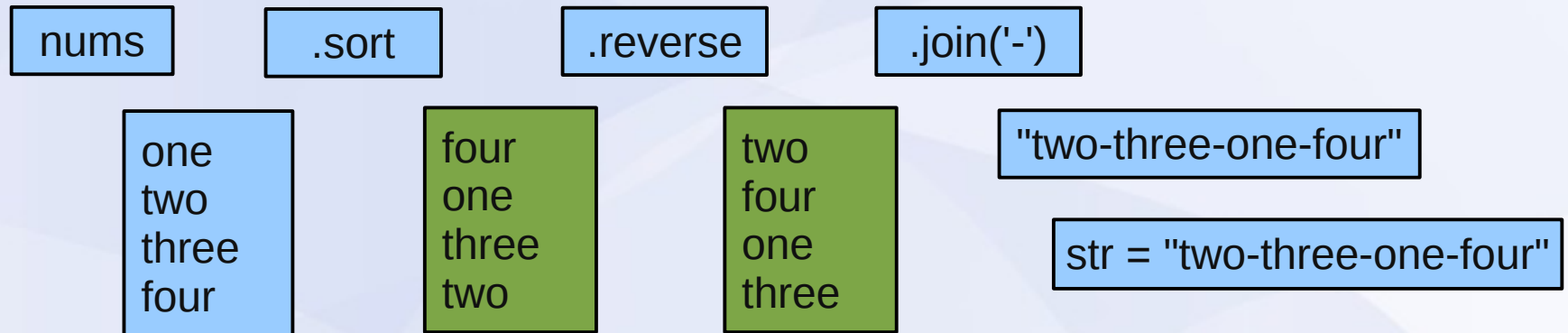


# Method chaining

## 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('-')
```

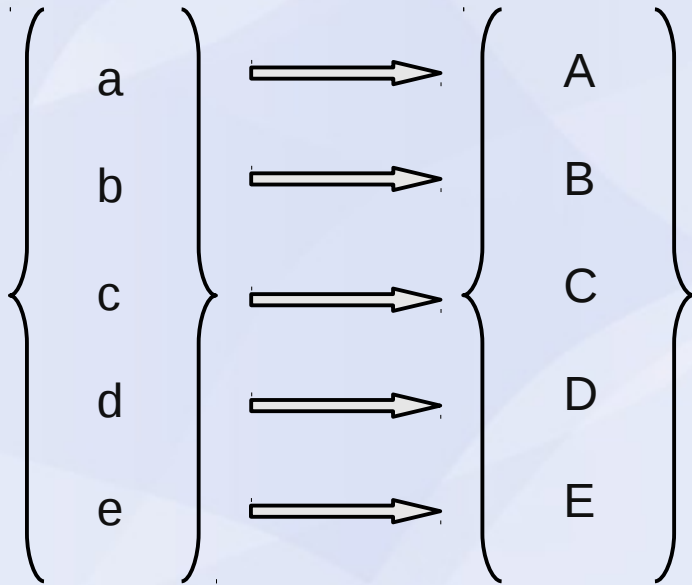
```
str = "two-three-one-four"
```

```
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  
ссылаются 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](http://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.downto(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](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`



# Модули

everything you always wanted to know  
but were afraid to ask

# Модуль

➤ Модуль - набор методов, сгруппированных по назначению и вынесенных в отдельный файл

♦ возможность повторного использования в разных скриптах

модуль Math

cos  
sin  
tan

модуль Church

cross  
sin  
prayer

♦ помещение метода в модуль позволяет иметь одноименные методы с разной функциональностью

# Определение и использование

- Коллекция методов, работающих с теггированным текстом

module Syn

syn.rb

```
def self.untag(tagged)
  tagged.gsub(/_[^_\\s]+/, "")
end
```

```
def self.unword(tagged)
  ...
end
```

end

module Syn

syn.rb

```
def Syn.untag(tagged)
  tagged.gsub(/_[^_\\s]+/, "")
end
```

```
def Syn.unword(tagged)
  ...
end
```

end

- Использование:

Syn.untag( 'runs\_VBZ' ) #=> "runs"

- позже о том, как сделать, чтобы работало вот так:

"runs\_VBZ".untag

# Подключение модуля

- Модуль необходимо загрузить (обычно вверху файла)

```
require 'filename'
```

```
ri Kernel#require
```

например:

```
require 'syn.rb'
```

```
ri Kernel#require_relative
```

или без расширения:

```
require 'syn'
```

в этом случае рубли будет искать имена  
`syn.rb`, `syn.so`, `syn.o`, `syn.dll`

см `lesson.14/modules/test_syn_module.rb`

```
require './syn'
```

не следует задавать относительные пути

```
require_relative 'syn'
```

поиск начнется с текущей директории

# Настоящее повторное использование

- Цель: сделать так, чтобы ruby мог найти файл с модулем

```
require "syn"
```

- В каких директориях require ищет файлы?

```
> ruby -e 'puts $:'
```

```
$:  
$LOAD_PATH
```

массив содержит  
пути, в которых ищет  
require

- Переменная окружения RUBYLIB (добавить в .bashrc)

```
export RUBYLIB=~/.lib:~/rufsl/lib:$RUBYLIB
```

После чего необходимо перезапустить  
терминал (сигвин) или выполнить команду

```
> source ~/.bashrc
```

без пробелов  
вокруг =

# Задание

- ♦ Создайте директорию для собственных модулей (например lib в домашней директории) и настройте сигвин так, чтобы эта директория была в RUBYLIB.

создать директорию:

```
> mkdir ~/lib
```

добавить в .bashrc:

```
export RUBYLIB=~/lib:$RUBYLIB
```

- ♦ Создайте (в ~/lib) модуль Syn (можно взять lesson.14/modules/syn.rb) и добавьте в него метод words, который принимает на вход теггированную строку и возвращает все слова (без тегов) как массив.

один из способов решения:  
использовать уже существующий  
в модуле метод untag

```
def self.words ts  
  untag(ts).split  
end
```

- ♦ Методы внутри модуля могут вызывать друг друга без явного указания имени модуля в качестве ресивера (т.е. не Syn.untag и просто untag)

# Домашнее задание

♦ Реализуйте следующие методы модуля Syn

- ♦ tags      принимает через аргумент теггированное предложение, возвращает все теги в виде массива
- ♦ unwdc    принимает через аргумент теггированное предложение, возвращает это же предложение, но с объединенными в одно слово компаундами. Все внутренние теги сложных слов начинаются на wdc (wdcEL, wdcSN, wdcSJ, wdcSV, wdcLK)

```
das_ATDNN H_wdcEL -_wdcLK Bomben_wdcEL versuch_NCNSN  
=>  
das_ATDNN H-Bombenversuch_NCNSN
```

- ♦ какие-нибудь другие методы, по желанию

По желанию, разработайте тесткейсы для этого модуля (см. дальше).  
Используйте шаблон из `lesson.14/modules/unit_test_stub.rb`

# Именованние модулей и файлов

- ♦ имя модуля (и класса) является константой рубли - должно начинаться с большой буквы
- ♦ CamelCase в имени модуля, snake\_case в имени файла

имя модуля	имя файла
Syn	syn.rb
MySuperSin	my_super_sin



# Модули-примеси (mixin)

- Сравните определения и способы вызова

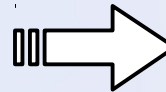
ЭТОТ МОДУЛЬ МОЖНО  
ПОДМЕШАТЬ К КЛАССУ

```
module Syn  
  def self.untag  
    ...  
  end  
end
```

```
module Syn  
  def untag  
    ...  
  end  
end
```

нет self

подмешанные методы  
становятся собственными  
методами объектов



```
class String  
  include Syn  
end
```

подмешивание  
к классу String

Использование:

```
Syn.untag("runs_VBZ")
```

```
"runs_VBZ".untag
```

см. [lesson.14/modules/syn\\_as\\_mixin.rb](#)

## + и -

- ◆ Плюсы: примеси позволяют легко добавить однотипную функциональность в несколько классов
- ◆ Минусы: примеси загрязняют стандартные классы
  - ◆ Что, если кто-то другой уже добавил в класс `String` метод `untag` с другим поведением?

# Вложенные модули

- Структурирование модулей (и классов) в сложной системе

```
module ESE
  module Tagger
    def self.method1
    end
  end
end
```

```
module Extractor
  def self.method1
  end
end
```

```
module ESE
  module Tagger
    def self.method1
    end
  end
end
```

ese/tagger.rb

```
module ESE
  module Extractor
    def self.method1
    end
  end
end
```

ese/extractor.rb

ИСПОЛЬЗОВАНИЕ:

ESE::Tagger::method1

fully-qualified name

ESE.Tagger.method1

# TDD с использованием Test::Unit

- ▶ см Unit::Test в lesson.14/modules/test\_quality.rb

```
class TestQuality < Test::Unit::TestCase
```

- ▶ Чтение:

[http://en.wikibooks.org/wiki/Ruby\\_Programming/Unit\\_testing](http://en.wikibooks.org/wiki/Ruby_Programming/Unit_testing)

# Задание

- Разработайте модуль Quality согласно тесткейсам, которые заданы в `lesson.14/modules/test_quality.rb`

ответ: `lesson.14/modules/quality.rb`

- По желанию добавьте в модуль что-нибудь свое, например, вычисление F-Measure.
- По желанию добавьте тесткейсы в файл с тестами `test_quality.rb`

# Hash

в питоне: Dictionary

# Hash

- Хэш (ассоциативный массив) *неупорядоченная (?)* коллекция пар "ключ-значение".
- Ключи в *массиве* (индексы) это целочисленные значения (от 0 и выше).

```
arr_months = [ "January", "February", "March" ]
```

```
arr = []
```

- Ключом в хэше может быть любой объект (строки, числа, символы, массивы...). Ключ должен быть уникальным.

не путать с  
%w{ ... }

```
hash_months = {  
  "Jan" => "January",  
  "Feb" => "February",  
  "Mar" => "March"  
}
```

```
hash_months = {  
  1 => "January",  
  2 => "February",  
  3 => "March",  
}
```

```
hsh = {}
```

```
ri Hash#[]
```

```
hash_months["Jan"] #=> "January"
```

```
hash_months[2] #=> "February"
```

- Значением в хэше (как и в массиве) может быть любой объект.

# Hash

- Сколько ключей в этом хэше?

```
hash = { 1 => 'one', "1" => 'uno' }
```

`#=>2`

```
ri Hash#length
```

- Еще один способ инициализации хэша, появился в ruby 1.9

```
numbers = { one: 'uno', two: 'dos', three: 'tres' }
```

- Какому классу принадлежат ключи хэша numbers?

см. `lesson.15/test_hash_with_colons`

классу `Symbol`

- Исследуйте, будет ли работать это?

```
hash = { 1: 'uno' }
```

```
hash = { "two": "dos" }
```

в обоих случаях  
ошибка



# Изменение хэша

- Метод []= добавляет или замещает пару ключ-значение

```
hash_months["Apr"] = "April"
```

ri Hash#[]=

```
hash_months ["May"] = "Can"
```

```
hash_months ["May"] = "May" ← останется только эта пара
```

ключи в хэше являются уникальными

- Синоним: метод Hash#store

```
hash_months.store("Jun", "June")
```

```
hash_months.store "Jul", "July"
```

ri Hash#store(key, value)

# Вопросы к Хэшу?

- Какой метод позволяет узнать, есть ли в хэше какие-нибудь данные?

Hash#empty?

hash\_months.empty?    #=>false

- Как узнать, есть ли в хэше некоторый **ключ**?

has\_key?(k)

key?(k)

include?(k)

member? (k)

hash\_months.key?("Jan")

#=> true

hash\_months.key?("January")

#=> false

- Как узнать, есть ли в хэше некоторое **значение**?

Hash#value?(v)

Hash#has\_value?(v)

hash\_months.value?("January")

#=> true

# Получение данных из хэша - 1

- Метод `[]` позволяет получить значение по указанному ключу

```
hash_months["Jan"] #=> "January"
```

ri Hash#[]

- Чтобы получить значения по нескольким ключам?

Hash#values\_at

поиск ключа происходит:  
1) с учетом регистра  
2) с учетом типа  
данных

- Что вернет следующая команда?

```
hash_months.values_at "Feb", "apple", "Jan"
```

```
#=> ["February", nil, "January"]
```

см. lesson.15/test\_values\_at

массив значений (nil, если ключа нет) в порядке, соответствующем порядку аргументов при вызове Hash#values\_at

- Исследуйте скрипт lesson.15/test\_values\_at.2

## Получение данных из хэша - 2

- ▶ Как получить список всех ключей, которые есть в хэше?

проверьте ваши идеи в irb используя данные из файла loadme

```
load 'loadme'  
@months.keys
```

```
Hash#keys
```

- ▶ в руби 1.8 порядок элементов в хэше неопределен

```
#=> ["Jul", "Apr", "Jan", "Feb", "Mar", "Jun", "May", ...] ← массив!
```

- ▶ в руби 1.9 – в порядке добавления элементов в хэш

```
#=> ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", ...] ← массив!
```

## Получение данных из хэша - 3

- ♦ Вопрос: как получить *все* значения, хранимые в хэше

Hash#values

@months.values

- ♦ в руби 1.9

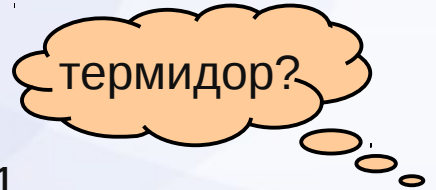
#=> ["January", "February", "March", "April", "May", "June", "July"]

- ♦ Значения в возвращаемом массиве упорядочены по тем же правилам, что и ключи (то есть, нет порядка в 1.8 и в порядке добавления в 1.9)

# Значение по умолчанию - 1

- Обращение к несуществующему ключу возвращает nil

```
hash_months ['term'] #=> nil
```



- Проблемная ситуация?

```
counts = {}  
counts['apple'] += 1
```

решение #1

```
counts = {}  
counts ['apple'] ||= 0  
counts ['apple'] += 1
```

- Решение #2 – метод Hash#default=

```
counts = {}  
counts.default = 0  
  
counts['apple'] += 1  
counts['grapes'] += 1
```

```
ri Hash#default=  
ri Hash#default
```

## Значение по умолчанию - 2

- Решение #3 – в момент инициализации можно указать значение по умолчанию

```
counts = Hash.new(0)

counts['apple'] += 1
counts['barmelely'] += 1
```

```
ri Hash.new
```

- Что будет напечатано в двух случаях?

```
data = Hash.new("hello")

puts data[9]
data[9].upcase!

puts data['ten']
```



это единственный объект, он  
присваивается всем новым ключам

```
#=> "hello"
```

```
#=> "HELLO"
```

# Мантра о новом дефолтном объекте

- При обращении к несуществующему ключу, будет создаваться пара

ключ => новый массив

```
hash = Hash.new { |h,k|  
  h[k] = []  
}
```

многоязычный словарь:

```
words = Hash.new { |h,k| h[k] = []}  
  
words['apple'] << 'Apfel' << 'manzana'  
words['butterfly'] << 'Schmetterling' << ...
```



# Итерирование по хэшу

- ♦ Хэш – **неупорядоченная (?)** коллекция.
  - в руби 1.8 – неупорядоченная
  - в руби 1.9 – упорядоченная в порядке добавления
- ♦ Какие есть итераторы в хэшах?

```
Hash#each  
Hash#each_pair  
  
Hash#each_key  
Hash#each_value
```

возвращает пару ключ-значение

```
hash_months.each do |abbr, full|  
  puts "#{abbr} means #{full}"  
end
```

- ♦ Будет ли это работать? одна переменная вместо двух для ключа и значения?

```
hash_months.each { |a|  
  puts a.inspect  
}
```

#=> ["Jan", "January"]

# Задания

- Есть ли какие-либо отличия в работе между следующими командами?

```
hash_months.each_key do | abbr |  
  puts abbr  
end
```

```
hash_months.keys.each do | abbr |  
  puts abbr  
end
```

# Задание

- Разработайте скрипт, который находит в списке data/words.txt палиндромы и вольвограммы. Игнорируйте слова длины 1.

палиндром: civic

вольвограмма: stun ↔ nuts

ожидаемый выход: find\_palindromes.out

(в lesson.15/tasks)

ответ: find\_palindromes.rb

- Сделайте вторую реализацию этого скрипта, но с использованием массива вместо хэша.

ответ: find\_palindromes\_over\_array.rb

Сравните время выполнения двух скриптов

```
> time -p find_palindromes.rb ...
```

```
> time -p find_palindromes_over_array.rb ...
```

Хэши быстрее!

# Задание

- ♦ Разработайте скрипт, который находит в списке слов data/words.txt анаграммы заданного слова (кроме самого заданного слова). Слово задается как аргумент при вызове скрипта. Скрипт должен игнорировать различия в регистре (cat и Act нужно считать анаграммами) но выводить слова в первоначальном регистре.

примеры запуска и файлы с ожидаемым выходом:

```
> find_anagrams_of Reward    # см. find_anagrams_of.Reward  
> find_anagrams_of resist    # см. find_anagrams_of.resist
```

начальный скрипт: find\_anagrams\_of\_stub (в lesson.15/tasks)

ответ: find\_anagrams\_of

## Домашнее задание

- Разработайте скрипт, который найдет в файле data/words.txt все анаграммы и выведет каждую группу слов в одну строку через табуляцию. Игнорируйте регистр написания при поиске анаграмм, но выводите слова в исходном регистре

```
whiter      \t wither \t writhe  
woodworm   \t wormwood
```

ожидаемый выход: find\_all\_anagrams.out

(в lesson.15/tasks)

ответ: find\_all\_anagrams

- Дополнительное задание (по желанию):

сделайте так, чтобы не считались анаграммами такие группы, где *все слова* суть разные регистровые написания одного слова, например:

```
Workman    \t workman
```

ожидаемый выход: find\_all\_anagrams.2.out

ответ: find\_all\_anagrams.2

# Массив ARGV

- Массив ARGV содержит все аргументы, с которыми вызывается скрипт, в том же порядке, в каком они указаны в командной строке.

```
> find_anagrams_of reward filename1 filename2
```

то массив ARGV содержит следующие **строки**

```
["reward", "filename1", "filename2"]
```

- Важно: "reward" не является файлом, поэтому попытка его читать вызовет ошибку

см. `lesson.16/test_argv.sh`

```
./test_argv:5:in `gets': No such file or directory - reward (Errno::ENOENT)
```

# ARGV.shift

- Аргументы не-файлы нужно удалить из массива ARGV

```
query = ARGV.shift
```

```
ri Array#shift
```

```
#=> query = "reward"
```

```
#=> ARGV = ["filename1", "filename2"]
```

- Другие возможности:

```
ARGV.delete_at(0)
```

- Почему не будет работать?

```
ARGV = ARGV[1..-1]
```

```
warning: already initialized constant ARGV
```

см. lesson.16/test\_argv\_2

это два разных массива ARGV

# File.readlines

- Метод класса File.readlines позволяет зачитать весь файл целиком в массив.

```
lines = File.readlines(fname)
```

```
ri File.readlines
```

зачитает с разделителем по умолчанию, то есть \n

- см. lesson.16/test\_file\_readlines

The file shortfile.txt contains the following 4 lines:

```
["warder is a volvogram\n", "\n", "deified is a palindrom\n", "\n"]
```

- Задание: попробуйте указать в скрипте test\_file\_readlines в качестве разделителя пустую строку ""

```
File.readlines(fname, "")
```

The file shortfile.txt contains the following 2 lines:

```
["warder is a volvogram\n\n", "deified is a palindrom\n\n"]
```



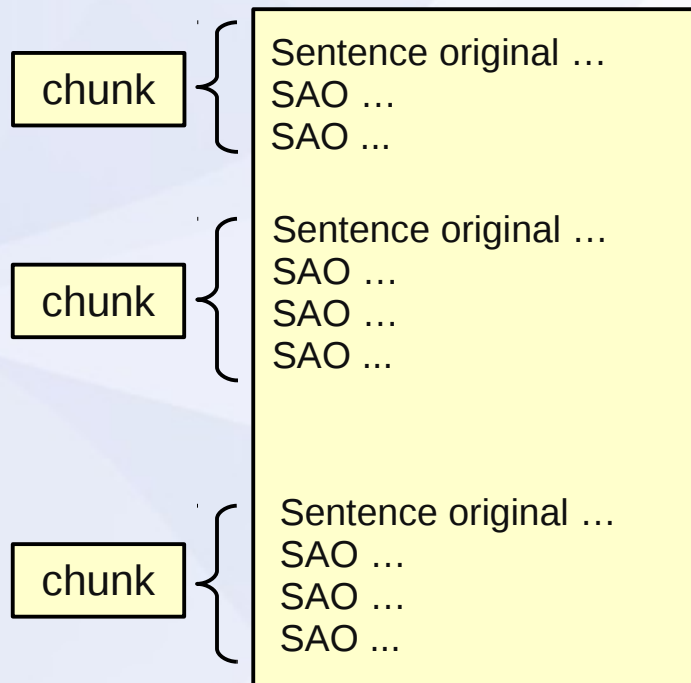
# Чтение текста блоками

- Разделитель “” позволяет читать файл фрагментами, разделенными как минимум одной пустой строкой.

```
chunks = File.readlines(fname, “”)
```

- Это так же справедливо для IO#gets

```
while chunk = gets(“”)  
  ...  
end
```



- Блок текста *chunk* является одной строкой, с \n внутри

# Инициализация Хэша из Массива

- Хэши быстрее массивов => лучше использовать хэши
- Массив массивов вида:

```
[ [k1, v1], [k2, v2], [k3, v3] ... ]
```

можно интерпретировать как массив пар ключ-значение и преобразовать в ХЭШ:

```
keys_and_values = [[1, 'one'], [2, 'two'], [3, 'san']]  
h = Hash[ keys_and_values ]
```

```
ri Hash.[]
```

```
#=> {1=>"one", 2=>"two", 3=>"san"}
```

см. [lesson.16/test\\_hash\\_from\\_array](#)

# Преобразование хэша в массив

- “Обратная” операция

```
h = {1=>"one", 2=>"two", 3=>"san"}
```

```
h.to_a
```

```
ri Hash#to_a
```

```
#=> [[1, 'one'], [2, 'two'], [3, 'san']]
```

- Преобразование в массив происходит при сортировке хэша

# Сортировка хэша

- ▶ Hash#sort, Hash#sort\_by – работают как в массивах, потому что оба происходят из модуля-примеси Enumerable, общего для Array и Hash
- ▶ Эти методы возвращают **массив**, потому что хэш не считается *упорядоченной* структурой данных.
- ▶ Выполните скрипт lesson.17/hashsort/test\_sort\_1.rb

```
[ ["0", "zero"], ["1", "one"], ["10", "ten"], ["11", "eleven"], ["12", "twelve"],  
  ["2", "two"], .... ]
```

```
@eng_numerals.sort.class #=> Array
```

- ▶ Вопрос: по какому принципу были упорядочены элементы хэша?

Ответ: в порядке увеличения (строкового) значения ключей

“0” < “1” < “10” < “11” < “12” < “2”

# Сортировка хэша с блоком

- Метод Hash#sort применяет вот такой блок по умолчанию:

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

```
["0", "zero"] <=> ["10", "ten"]
```

```
hash = {  
  "0" => "zero",  
  "1" => "one",  
  "2" => "two"  
}
```

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

- Выполните и проанализируйте скрипт hashsort/test\_sort\_2.rb

фрагмент вывода:

```
a=["12", "twelve"]  
b=["5", "five"]  
result of comparison: -1
```

35 сравнений для  
сортировки 13 элементов

```
[..., ["12", "twelve"], ["2", "two"], ["3", "three"], ["4", "four"], ["5", "five"], ...]
```

# Задания

- Измените test\_sort\_2.rb так, чтобы хэш был отсортирован в *нисходящем* порядке по *численному* значению ключа.

ожидаемый выход:

```
[["12", "twelve"], ["11", "eleven"], ["10", "ten"], ["9", "nine"], ["8", "eight"],  
["7", "seven"], ["6", "six"], ["5", "five"], ["4", "four"], ["3", "three"],  
["2", "two"], ["1", "one"], ["0", "zero"]]
```

ответ: hashsort/test\_sort\_3.rb

- Отсортируйте хэш @numerals (задан в файле data) по испанским числительным в порядке возрастания (в алфавитном порядке).

ожидаемый выход:

```
[["0", ["zero", "ceero"]], ["5", ["five", "cinco"]], ["4", ["four", "cuatro"]], ["10", ["ten",  
"diez"]], ["12", ["twelve", "doce"]], ["2", ["two", "dos"]], ["9", ["nine", "nueve"]], ...]
```

ответ: hashsort/test\_sort\_4.rb

# Sort!ировка хэша

- Вопрос: почему нет методов *Hash#sort!* и *Hash#sort\_by!* ?
- Ответ: метод *Hash#sort* меняет сущность объекта: результат сортировки -- объект класса *Array*, а не *Hash*.

# Сортировка Hash#sort\_by

- Метод Hash#sort\_by позволяет избавиться от явного сравнения элементов (и оператора <=>).
- Вместо <=>, достаточно сделать так, чтобы блок возвращал то значение, по которому необходимо произвести сортировку.

```
@eng_numerals.sort_by do |key, val|  
  key.to_i  
end
```

см. hashsort/test\_sort\_5.rb

```
@eng_numerals = {  
  "0" => "zero",  
  "1" => "one",  
  "2" => "two",  
  "3" => "three",  
  "11" => "eleven"  
}
```

```
[["0", "zero"], ["1", "one"], ["2", "two"], ["3", "three"], ["4", "four"], ["5", "five"],  
["6", "six"], ["7", "seven"], ["8", "eight"], ["9", "nine"], ["10", "ten"], ["11",  
"eleven"], ["12", "twelve"]]
```

- Вопрос: как изменить порядок сортировки на нисходящий?

```
-key.to_i
```



# Задание

♦ Отсортируйте хэш @numerals (задан в файле data) по *обратному* чтению английских числительных (в порядке возрастания). Используйте Hash#sort\_by

ожидаемый выход:

```
[["3", ["three", "tres"]], ["9", ["nine", "nueve"]], ["1", ["one", "uno"]], ["5", ["five", "cinco"]], ["12", ["twelve", "doce"]], ["10", ["ten", "diez"]], ["11", ["eleven", "once"]], ["7", ["seven", "siete"]], ["0", ["zero", "cero"]], ["2", ["two", "dos"]], ["4", ["four", "cuatro"]], ["8", ["eight", "ocho"]], ["6", ["six", "seis"]]]
```

так как: ee < eni < eno < evi ...

ответ: hashsort/test\_sort\_6.rb

# Метод Hash#sort\_by\_value

- Такого метода нет
- Но его можно реализовать!

```
class Hash
  def sort_by_value
    self.sort_by { |k,v| v }
  end
end
```

- Использование (см. hashsort/test\_sort\_7.rb):

```
puts @eng_numerals.sort_by_value
```

```
[["8", "eight"], ["11", "eleven"], ["5", "five"], ["4", "four"], ["9", "nine"], ["1", "one"],  
["7", "seven"], ["6", "six"], ["10", "ten"], ["3", "three"], ["12", "twelve"], ["2", "two"],  
["0", "zero"]]
```

# Задание cmpsort

Полная формулировка в [lesson.17/cmpsort/README](#)

- ♦ Необходимо разработать скрипт (`cmp_ccs_sort_by_tagseq`), который переупорядочивает разницу по CCSplitter-y/MainWordExtractor-y так, чтобы ее было удобно тестировать:
  - ★ CCSplitter опирается на теги входной цепочки => сгруппировать записи разницы по входной цепочке
  - ★ Есть более частотные явления, есть менее частотные. Эту информацию можно учитывать, решая, что тестировать и что нет.

Входной файл:

английский: `cmp_ccsplitter_1.out`

немецкий: `german/cmp_ccsplitter_1.out`

Ожидаемый выход в: `sorted`

Ответ: `cmp_ccs_sort_by_tagseq`

# Задание find\_words

полное и пошаговое описание в `lesson.16/find_words/README`

- ♦ Разработать скрипт, который находит в текстовых файлах указанные слова (или фразы) и выводит их в формате

слово \tab позиция\_в\_предложении \tab предложение

- ★ Если указана опция `-i` или `--ignore-case`, то поиск осуществляется без учета регистра и вывод имеет вид:

СЛОВО \tab позиция\_в\_предложении \tab предложение

- ★ Об опции `-t` или `--output-totals` читайте в README

- ★ Файл со словами/фразами для поиска передается как первый аргумент в командной строке. Все остальные аргументы считаются текстовыми файлами, в которых происходит поиск.

```
> find_words queries.txt ../../data/corpus.tok.txt
```

- ★ Задание необходимо выполнять пошагово, как описано в README

# CRF

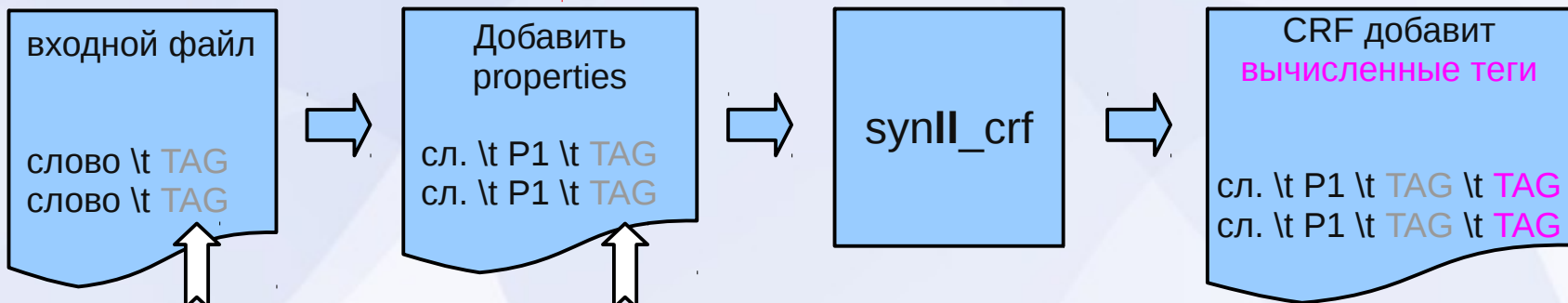
**ВАЖНИССИМО**

при назначении properties нельзя  
опираться на эталонный тег

## ➤ тренировка



## ➤ ИСПОЛЬЗОВАНИЕ



здесь теги не нужны – synll\_crf их не использует  
но умеет их игнорировать

config file

# Contest: textspacer/пробелизатор

➤ Реализовать при помощи CRF и других известных и неизвестных механизмов вставку пробелов в текст на русском языке, из которого были удалены пробелы.

для решения задачи не требуется **полностью токенизировать** текст (т.е. отделять знаки препинания), достаточно получить обычный книжный текст.

все материалы в `lesson.22/text_spacer`

о работе с UTF-8 см. дальше, в разделе Strings

вместо больших файлов `train.txt` и `test.txt` (которые необходимо использовать для тренировки и тестирования рабочей модели CRF) для разработки вспомогательных скриптов нужно использовать файлы `train_short.txt` и `test_short.txt`

Изучить `test_short.txt` и `train_short.txt`

# textspacer: выбор таргета

♦ Таргет должен быть таким, чтобы он позволял выполнить задачу - вставить пробелы. Теги должны сообщать, как вставляется пробел(ы) относительно данного символа.

## ♦ Таргет #1

SL	space on the <b>l</b> eft hand side
SR	space on the <b>r</b> ight hand side
SB	space on <b>b</b> oth sides

## ♦ Таргет #2

SI	<b>i</b> nitial symbol
SM	<b>m</b> iddle symbol
SL	<b>l</b> ast symbol
SW	<b>s</b> ymbol is word

♦ Задание: заполните необходимые секции в конфигурационном файле `crf_text_spacer_stub.cfg`

описание: <http://syn-proc5/wiki/crf/>



# textspaser: подготовка корпусов - 1

♦ Требования к формату вытекают из того, что

1. реальные данные приходят в том виде, как дано в **тестировочном** корпусе – исходим из тестировочного корпуса

i. **минимальной** единицей входа для CRF является **один СИМВОЛ**

ii. в тексте есть **параграфы** – нужно (?) их сохранить

iii. *именно для этой задачи, также необходимо сохранить первоначальное количество пустых строк между параграфами*

2. тренировочный и тестировочный корпуса должны быть в одном формате, но надо учесть, что:

i. если в тестировочном корпусе присутствует эталонный тег, то synll\_crf его игнорирует.

ii. если в тестировочном корпусе тега нет, то строка должна заканчиваться на табуляцию (проверено на данных с одним полем)



## textspaser: подготовка корпусов - 2

### 3. формат, понятный утилитам `syn{ll,b}_crf`

как в `assign_properties`

слово \tab тег

слово \tab тег

*[между параграфами – одна или более пустых строк]*

слово \tab тег

слово \tab тег

i. лишние пустые строки игнорируются

ii. при выводе в `synll_crf`, между записями вставляется **только одна** пустая строка

противоречие с п. 1-iii

Решение: использовать механизм `–skip-line`, чтобы протянуть пустые строки через `synll_crf`

# textspaser: подготовка корпусов - 3

## ➤ Необходимые действия

	train.txt	test.txt
1. вертикализировать	да	да
✶ 2. добавить теги	да	нет (невозможно)
3. удалить пробелы	да	нет/да (пробелов там нет)
4. замаскировать пустые строки	нет/да	да

## ➤ Отличия невелики – стоит сделать один скрипт для обоих корпусов

- Вопрос: стоит ли ввести опцию, для различения видов корпусов или реализовать угадывание в скрипте?

Дихлордиметилтрихлорметилметан!



Первая строка в корпусе

## textspaser: подготовка корпусов - 4

- ♦ Задание: реализуйте скрипт `verticalize`, преобразующий исходные файлы `train.txt` и `test.txt` в необходимый формат. Сделайте опцию для добавления тегов. Убедитесь, что работает опция `--label-empty-lines`.

п4 таблицы (маскировка пустых строк) уже реализована через опцию `--label-empty-lines`, по которой вместо пустых строк исходного файла вставляется строка `###EMPTYLINE`.

начальный скрипт: `verticalize_stub`  
возможная реализация: `verticalize`

Ожидаемый выход:

```
test_short.txt -> test_short.vert  
train_short.txt -> train_short.vert
```

# textspaser: конфигурирование CRF

- Сам по себе символ уже является property, и из него можно создать ряд фич
- Общий вид шаблона для **униграммных** фич

```
UG \tab NAME \tab LINENUM, FIELDNUM [\tab LINENUM, FIELDNUM]
```

где:

LINENUM = 0 указывает на текущую строку

LINENUM > 0 указывает на последующие строки

LINENUM < 0 указывает на предшествующие строки

# current character text

UG CW 0,0

# previous character text

UG PW -1,0

# previous character combined with the current character

UG PCW -1,0 0,0

	CW	PW	PCW
К	К	#	#К
Н	Н	К	КН
И	И	Н	НИ
Г	Г ...		
а	а ...		

# textspacer: конфигурирование

♦ Задание: добавьте шаблоны для генерации следующих фич

- current character
- previous character
- next character
- previous character + current character
- current character + next character
- previous character + current character + next character

♦ Вопрос: какие значения должны иметь переменные

STR\_WORD\_FIELDS\_COUNT = 1  
INT\_WORD\_FIELDS\_COUNT = 0

Потому что в файле, идущем на вход synb\_crf, только одно поле (кроме тега) и оно строковое

♦ Задание: включите бинарную фичу, которая бы использовались соседние теги.

ответ: crf\_text\_spacer.crf

# textspacer: назначение других properties

- Если нужны дополнительные свойства, то их можно добавить в файл, который идет на вход утилитам CRF

```
слово \tab PROP1 \tab PROP2 \tab тер  
слово \tab PROP1 \tab PROP2 \tab тер
```

- ❖ Важно: Количество полей должно во всех строках совпадать!
- При этом в конфигурационном файле необходимо:
  - Изменить значения переменных 

```
STR_WORD_FIELDS_COUNT  
INT_WORD_FIELDS_COUNT
```
  - Добавить шаблоны для преобразования новых свойств в фичи
- Задание: используйте скрипт **assign\_properties**, чтобы добавить как отдельное свойство текст символа в нижнем регистре.

см. дальше, как работать с регистром в UTF-8

# textspacer: тренировка модели

- Исходник корпуса (статичен)

К	SI
н	SM
и	SM
г	SM
а	SL

- Обогащен доп. propertyями (изменяется)

К	P1	SI
н	P1	SM
и	P1	SM
г	P1	SM
а	P1	SL

synb\_crf

- Как запускать:

```
cat train.vert |  
assign_properties |  
synb_crf --bin=model.bin --min-feat=2 --config=crf_text_spacer.cfg
```

Скрипт assign\_properties не нужен, если дополнительные property не используются

train\_crf

- Пример промежуточной информации, выводимой synb\_crf: *synb\_crf.log*  
Exit code of training procedure: 1 – это хорошо



# textspacer: keep your stuff under control

♦ Используйте git для хранения версий нужных файлов: *assign\_properties*, *crf\_text\_spacer.cfg*

♦ чтобы создать репозиторий в текущей директории

```
git init .
```

♦ чтобы добавить к проекту файлы

```
git add file1 file2...
```

♦ чтобы закоммитить в *локальный* репозиторий

```
git commit -m "message"
```

♦ пушить в *удаленный* репозиторий не надо (его нет, но можно настроить удаленный репозиторий)



# textspaser: применение в боевых условиях

- Чтобы использовать модель для теггирования:

```
cat train.vert | assign_properties | synll_crf --bin=model.bin --stat
```

Файл `train.vert` содержит эталонные теги, опция `--stat` позволяет получить цифры по качеству работы теггера, сравнивая эталонные и вычисленные теги.

- Опция `--skip-line=XXX` позволяет пропустить через утилиту некоторую строку и получить ее в неизмененном виде в выдаче.

По условию соревнования в выдаче необходимо сохранить пустые строки между параграфами, которые были замаскированы при помощи метки `#!#EMPTYLINE`.

```
cat test.vert | assign_properties |  
synll_crf --bin=model.bin --skip-line='#!#EMPTYLINE'
```

# textspaser: вставка пробелов

- ♦ Задание: разработайте скрипт, вставляющий пробелы в текст согласно тега, вычисленной моделью.

Скрипт должен уметь принимать данные с любым количеством полей:

```
restore_spaces.1.in  
restore_spaces.2.in
```

Ожидаемых выход для обоих входных файлов одинаковый:

```
restore_spaces.all.out
```

Скрипт должен распознавать метку `###EMPTYLINE` и вставлять вместо нее пустую строку. Эта логика уже реализована.

Начальный скрипт: `restore_spaces_stub`

Возможная реализация: `restore_spaces`

Congratulations! You have completed the quest.  
Now you can submit the test.txt file with restored spaces to the contest by  
copying it into predefined directory.

# String

строковый класс

# Charset

- ♦ Вначале было Слово и было оно класса String.
- ♦ Но еще раньше были числа.
- ♦ Любому символу из кодовой страницы соответствует число, в однобайтной кодировке это число в диапазоне 0..255

<http://en.wikipedia.org/wiki/Windows-1251>      кириллическая

<http://en.wikipedia.org/wiki/Windows-1252>      латинская

- ♦ Символы с кодами 0..127 одинаковые. Это 7-битное ASCII
- ♦ Верхняя часть таблицы - языкозависимая.

код 196	Windows-1251	Д
	Windows-1252	Ä

# Порядок символов и локаль

- Локаль влияет на порядок сортировки символов

unsorted

A  
b  
B  
a  
Ö  
á

C

A  
B  
a  
b  
Ö  
á

ru\_RU.UTF8

a  
A  
á  
b  
B  
Ö

файл: lesson.21/chars.txt

- Give it a try

```
> cat chars.txt | LC_ALL=C sort
```

```
> cat chars.txt | sort
```

- Сортировка согласно локали C располагает символы в том порядке, в каком они расположены в кодовой таблице, то есть, в порядке возрастания их кодов.
- Сортировка согласно др. локалей опирается на другой порядок расположения символов.

# Символ <--> Код

- Преобразование символа в код и наоборот.

String#ord

"A".ord ==> 65  
"a".ord ==> 97

Integer#chr

65.chr ==> "A"  
97.chr ==> "a"

- Вопрос: какой код имеют символы пробел, табуляция, \n, \r и пустая строка?

" ".ord ==> 32  
"\t".ord ==> 9  
"\n".ord ==> 10  
"\r".ord ==> 13  
"".ord



пустая строка  
не является символом

- Вопрос: какой код имеет символ Ö и в какой кодировке его видит Руби?

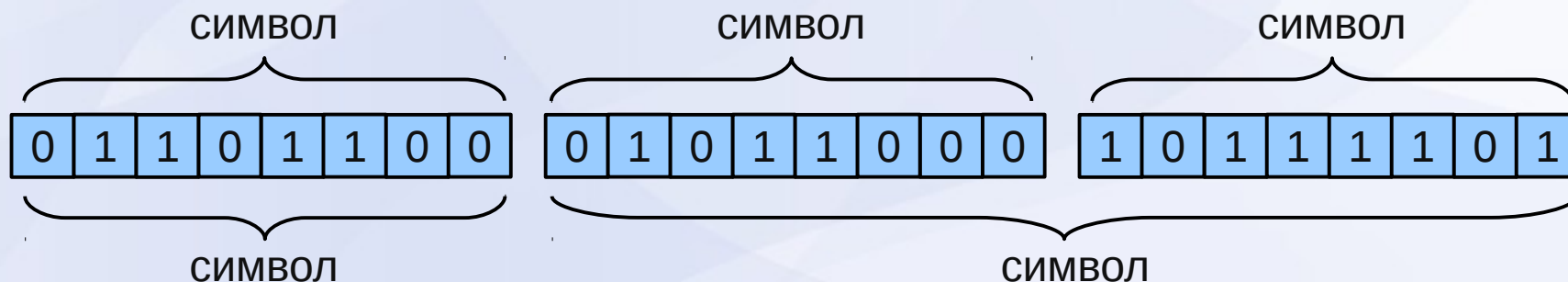
"Ö".ord ==> 214, как в windows-1252  
"Ö".encoding ==> #<Encoding:UTF-8>

ri String#encoding

# UTF-8

## ♦ Однобайтные vs. многобайтные кодировки

♦ однобайтные кодировки: 1 байт = 1 символ



♦ Многобайтные кодировки: 1+ байт (октетов) = 1 символ

♦ Исследуйте, как работает скрипт `lesson.22/unicode/test_russian_utf8` без установленной внешней кодировки и с ней

```
Encoding::default_external = 'UTF-8'
```

Разное разбиение,  
разная длина символов

```
> test_russian_utf8 test_russian_utf8.txt
```



# Преобразование регистра в UTF-8

- Исследуйте скрипт `unicode/test_russian_utf8_2` с `@use_mb_chars=true|false`  
`> test_russian_utf8_2 test_russian_utf8.txt`    • Регистр не преобразуется!

- Регистровые преобразования не поддерживаются в ядре языка

- потому что регистровые преобразования зависят от локали

В турецком:	$I \Leftrightarrow I$	dotted/undotted i
i с точкой и без	$i \Leftrightarrow İ$	<a href="http://en.wikipedia.org/wiki/Turkish_alphabet">http://en.wikipedia.org/wiki/Turkish_alphabet</a>

- некоторые преобразования не биективны (не взаимно-однозначны)

в немецком:  $\text{ß} \Rightarrow \text{SS}$ , но не всякое  $\text{SS} \Rightarrow \text{ß}$

- может быть, есть языки, в которых прописные выглядят как строчные. То есть, нельзя однозначно приписать символу свойство lowercase или uppercase



# Преобразование регистра: решение

- Использовать библиотеки
  - механизм Multibyte из ActiveSupport

```
shell> gem install active_support i18n
```

установка

```
require 'active_support/lazy_load_hooks'  
require 'active_support/core_ext/string'
```

# may not be necessary

```
'ящерица'.mb_chars.upcase.to_s
```

#=> "ЯЩЕРИЦА"

- gem (пакет) unicode\_utils (<http://unicode-utils.rubyforge.org/>)

```
shell> gem install unicode_utils
```

```
require 'unicode_utils'
```

```
UnicodeUtils.upcase('ящерица')
```

#=> "ЯЩЕРИЦ"

# Установка гемов простыми смертными

- По умолчанию пакетный менеджер gem устанавливает пакеты в системные директории (/usr/)

- в линуксе это может сделать только суперпользователь

- Чтобы установить в HOME:

```
> gem1.9.1 install --user-install active_support i18n
```

- настроить окружение (.bashrc) и перестартовать терминал

```
if which ruby >/dev/null && which gem >/dev/null; then  
  PATH="$(ruby -rubygems -e 'puts Gem.user_dir')/bin:$PATH"  
fi
```

~/.bashrc

- эта команда добавляет в переменную PATH новый путь

```
>echo $(ruby -rubygems -e 'puts Gem.user_dir'  
#=> /home/krot/.gem/ruby/1.9.1
```

# Неожиданный String#[]

см. lesson.23/test\_getting\_chars

# **Класс Ю**

## **ВЫЗОВ ВНЕШНИХ КОМАНД**

# Механизмы запуска внешних команд

- ▶ выполнение внешних команд/пайпов внутри скрипта на Руби
- ▶ выбор конкретного механизма зависит от потребностей
  - ▶ нужен только STDOUT или также и STDERR или оба потока, раздельно?
  - ▶ как много данных выдает запускаемая команда?
  - ▶ нужно ли вычитывать данные, когда процесс еще выполняется, или можно дождаться, когда команда отработает полностью?
  - ▶ нужны ли коды возврата выполняемых команд?
  - ▶ нужна ли возможность контролировать (напр. убить) запущенный процесс?

`...`

system()

IO.popen

IO.pipe

IO.select

Kernel#exec

Kernel#fork

## (Бесполезный) `exec`

- `exec` - создает новый процесс для выполняемой внешней команды и *замещает* им текущий

```
exec "ls -l | wc -l"
```

см. [lesson.23/test\\_exec](#)

- выполнять можно как одиночную команду, так и их последовательность или конвейер
- Если выполнить эту команду в `irb`, то произойдет выход из `irb`-а
- Команда `exec` есть и в `shell/bash`

# Kernel#system()

- ♦ system -- как и exes, но команды выполняются в subshell-е. Как и в случае с exes, нет возможности получить данные, выведенные командой на STDOUT.

```
system "ls -l | wc -l"
```

см. lesson.23/test\_system

```
system(cmd) есть и в AWK
```

- ♦ плюсы:

- ♦ можно использовать в условиях, так как system возвращает true, если команда была выполнена успешно.
- ♦ автоматически устанавливается переменная \$? (объект класса Process::Status), содержащая информацию о последней выполненной внешней команде.

```
ri Process::Status
```

- ♦ Отличия \$? в баше и в руби

- ♦ в баше в этой переменной число - код возврата, любая команда устанавливает \$?
- ♦ в руби \$? это объект Process::Status, устанавливается вследствие выполнения только команды system(), `...` и др.

# Backticks (`)

- ▶ Backticks (`) -- выполняет внешнюю команду и возвращает ее STDOUT. Устанавливает \$?

```
today = `date`
```

```
#=> "Thu Jan 32 12:20:61 UTC 2034"
```

см. lesson.23/test\_backticks

- ▶ Надо знать:

- ◆ нет возможности многократно давать данные на STDIN
- ◆ нельзя использовать как пайп
- ◆ возвращается сразу весь STDOUT
- ◆ можно использовать интерполяцию #{ ... }

- ▶ Удобно использовать для разовых действий

```
ts = `echo Der Tisch steht da | TextTagger`
```

```
ri Kernel#`
```

также и в баше:

```
now=`date`  
echo $now
```

см. lesson.23/test\_backticks\_2



# Команда %x{ ... }

- Встроенная команда %x{ ... }. Работает аналогично Kernel#`

```
%x{ echo tags macht | syn_ldb --output-data }
```

см. lesson.23/test\_x

```
#=> macht VLPI2P VLPI3S VLPM2P NCFSN NCFSG NCFSD NCFSA
```

- Можно использовать любой парный знак препинания { } [ ] ( ) <>

♦ не путать с %w{ ... } для задания массивов строк

- Просто пример:  
(с интерполяцией)

```
if lang == "Japanese"  
  cmd = "echo #{text} | syn_rs2ts_jap"  
else  
  cmd = "echo #{text} | TextTagger"  
end  
  
res = %x[ #{cmd} ]
```

# IO.popen

♦ IO.popen - выполняет команду в дочернем процессе. Команда работает как пайп: в нее можно писать и читать одновременно.

♦ Изучите скрипт test\_popen\_1

```
cmd = IO.popen("./rev", "w+")  
cmd.puts "hello"  
cmd.gets  
cmd.close
```

```
ri IO.popen  
ri IO.puts  
ri IO.gets  
ri IO.flush
```

♦ Неизбежное зло - буферизация. Данные накапливаются в буфере, вместо того, чтобы быть распечатанными на STDOUT

♦ Задание: исследуйте, что произойдет, если в скрипте rev убрать строку:

```
STDOUT.flush
```

♦ Многие утилиты просао имеют опцию --flush для принудительного сброса буфера --> данные выводятся сразу.

# Don't break my pipe

- Задание: Исследуйте скрипт `test_popen_2`. Что происходит, если раскомментировать команду `exit` в скрипте `repeat`?

```
./test_popen_2:9:in `write': Broken pipe (Errno::EPIPE)  
from ./test_popen_2:9:in `puts'  
from ./test_popen_2:9:in `'
```

попытка дать данные на вход команде, которая уже завершилась,

- Хорошим тоном считается закрывать явно открытые объекты IO

```
IO#close
```

# IO.popen с блоком

➤ аргументом блока становится созданный объект класса IO

➤ Что напечатает эта команда?

Ответ: сегодняшнюю дату

```
IO.popen("date") { |io|  
  puts io.gets  
}
```

➤ Эта команда аналогична

```
puts `date`
```

```
io = IO.popen("date")  
puts io.gets  
io.close
```

➤ Блочная форма не требует явно закрывать созданный объект IO. Он закрывается сам, когда блок заканчивается.

# IO.popen с блоком

♦ Вопрос: что напечатает следующая команда и сколько будет напечатано строк?

```
IO.popen("ls -l") do |cmd|  
  puts cmd.gets  
end
```

см lesson.23/test\_popen\_3

Ответ: одну строку (IO#gets читает только одну строку).

♦ см. lesson.23/test\_popen\_4, вариант написания lesson.23/test\_popen\_2

# Режимы открывания файлов

➤ Сравните синтаксис:

```
cmd = IO.popen("date")
```

# DEFAULT: mode="r"

```
cmd = IO.popen("./rev", "w+")
```

mode="w+"

➤ Второй аргумент это IO open mode, задает режим на чтение или на запись/дописывание. Также влияет на то, произойдет ли ошибка, если файла/команды нет.

r	открывает на чтение, ругается, если файла нет
r+	открывает на чтение и запись

см. `lesson.23/test_popen_5`. Сравните режимы "r" или "r+" в `File.new`

No such file or directory - file\_that\_never\_existed\_5 (Errno::ENOENT) 198

# Режимы открывания файла

w открывает на запись, truncates to zero length, создает новый файл, если файл не существовал

w+ открывает на чтение и запись

см. lesson.23/test\_popen\_6: File.new и режим "w"

`gets': not opened for reading (IOError)

➤ Вопрос: что, надо сделать, чтобы ошибка пропала?

Ответ: использовать режим w+

a и a+ аналогично w и w+, но не обнуляет файл, а дописывает в конец

➤ Задание: исследуйте, какие из режимов открытия пайпа будут работать?

используйте скрипт: lesson.23/test\_open\_7\_stub

ответ: lesson.23/test\_open\_7

Tip of the day: используйте универсальный режим w+

# Открытие файла с указанием кодировки

- Из ri File.new

```
File.new(filename, mode="r" [, opt])
```

opt – это хэш

- Параметр opt может использоваться, чтобы задавать кодировки (внешнюю и внутреннюю), тип данных (текстовый/бинарный) и даже символ перевода строки

```
file = File.new(fname, "r",  
  external_encoding: 'UTF-8',  
  internal_encoding: 'Windows-1251')
```

Или еще короче

```
File.new(fname, "r:UTF-8:Windows-1251")
```

Данные зачитываются в utf-8, преобразуются в windows-1251, внутри скрипта манипулируются и выводятся в windows-1251

см. [lesson.23/test\\_file\\_new\\_enc](http://lesson.23/test_file_new_enc)



# Задание переменных окружения (ruby 2.0)

- Из `ri IO.popen`

```
IO.popen( [env,] cmd, mode="r" [, opt] )
```

- Команду можно запустить, установив ей окружение через параметр **env**, который представляет из себя **Hash**

```
env = { 'LDBPATH' => '/path/to/ldb-English' }  
tagger = IO.popen(env, "TextTagger", "w+")  
  
tagger.puts "The book is interesting"  
tagger.flush
```

- Демо на другом примере  
см. [lesson.24/test\\_open\\_9](#)

- Команда может быть дана с опциями
  - в форме одной строки

```
tagger = IO.popen(env, "TextTagger --flush --social", "w+")
```

- в форме массива (имя команды и опции – все отдельные элементы)

```
tagger = IO.popen(env, [ "TextTagger", "--flush", "--social" ], "w+")
```

# Задание переменных окружения в 1.9+

- Из `ri IO.popen`

```
IO.popen( cmd, mode="r" [, opt] )
```

- В ruby 1.9 отдельного параметра **env** нет, он является частью параметра **cmd**, который в этом случае задается как массив **Array**

```
IO.popen( [ { 'LDBPATH' => '/path/to/bin/ldb' }, "TextTagger" ], "w+")
```

то же, но пошагово

```
env = { 'LDBPATH' => '/path/to/bin/ldb' }  
cmd = [ env, "TextTagger", "--flush", "--hmm-tagger" ]  
tagger = IO.popen(cmd, "w+")
```

NB! имя команды и  
опции даются как  
отдельные элементы  
массива

- Демо на другом примере см. `lesson.23/test_open_9`

## Еще один пример (ruby 1.9 и 2.0)

♦ Итого, чтобы запустить команду с опциями

см. lesson.24/test\_popen\_10\_extractor  
см. lesson.24/test\_popen\_11\_pipe

♦ вся команда с опциями может быть задана одной строкой

```
cmd = "EXTester2 --relations=r__VerbPhrase --line --separate-sentences"
```

```
extractor = IO.popen(cmd, "w+")
```

работает также для пайпов

♦ вся команда с опциями может быть задана массивом

```
cmd = ["EXTester2", "--relations=r__VerbPhrase", "--line", "--separate-sentences"]
```

```
extractor = IO.popen(cmd, "w+")
```

не работает для пайпов

♦ способ через массив позволяет дополнительно установить переменные окружения (в виде хэша)

```
env = {....}
```

```
cmd = [env, "EXTester2", "--relations=r__VerbPhrase", "--line", '--flush']
```

```
extractor = IO.popen(cmd, "w+")
```

не работает для пайпов

# How to IO#gets

- ♦ Как читать выход команды, если неизвестно, сколько строк она возвращает на единицу входа?

```
cmd.puts "hello"  
line = cmd.gets
```

здесь все предсказуемо:  
одна строка входа → одна строка выхода

- ♦ см. lesson.24/clever\_echo

выводит каждую зачитанную строку столько раз, сколько символов в строке, завершая вывод пустой строкой

- ♦ Задание: исследуйте, что происходит, если выполнить скрипт lesson.24/test\_popen\_8\_hangs, использующий внутри пайп с clever\_echo

```
> test_popen_8_hangs words.txt
```

Ответ: скрипт зависает на попытке читать в цикле (gets) то, что *clever\_echo* печатает в ответ на **первую** строку ввода.

# How to IO#gets

♦ Задание: решите проблему, обозначенную в lesson.24/test\_popen\_8\_hangs

Ответ: lesson.24/test\_popen\_8

• Необходимо найти признак, указывающий на окончание вывода:

♦ некоторые утилиты просао завершают вывод пустой строкой;

♦ многие утилиты просао поддерживают опцию **--skip-line**

через **--skip-line** можно передать утилите какую-нибудь строку-маркер, и потом вылавливать эту строку в выходе этой утилиты

```
( echo "Want a (sugary) cookie?"; echo "ENDOFSENTENCE";  
echo "Yes, please" ) | TextTagger --skip-line=ENDOFSENTENCE
```

ВЫХОД:

Want_VB	a_AT	cookie_NN	?_?	} 1-to-2
sugary_JJ				
ENDOFSENTENCE				
Yes_UH	,_	please_VB		

см. также  
echo -e "a \n b"

# Задание

- ♦ Выясните, можно ли использовать `IO#read`, чтобы читать данные из команды, запущенной при помощи `IO.popen`

используйте `ri` и, например, скрипт `lesson.24/clever_echo`

Ответ: из `ri IO#read`:

If length is omitted or is nil, it reads until EOF...

состояние EOF (end-of-file) наступает, когда пайп закрывается

# Новое поколение выбирает Open3/Open4

- Библиотеки open3 и open4 позволяют более гибко контролировать stdout/stderr/stdin

```
ri Open3.popen3
```

- open4 распространяется в виде гема. Чтобы установить

```
gem search -r open4  
gem install open4
```

Использование:

```
require 'open4'  
Open4.popen4(...)
```

# **Regex & MatchData**



# Вспомнить все

- ▶ Регулярные выражения (РВ) можно метчить двумя способами

- ▶ Используя метод `String#=~` или `Regexp#=~`

```
str =~ re  
re =~ str
```

`#=>` возвращается число или `nil`

- ▶ Используя метод `String#match` или `Regexp#match`

```
str.match re  
re.match str
```

`#=>` возвращается объект класса `MatchData`

- ▶ Объект класса `MatchData` инкапсулирует информацию о метче.

# Вспомнить еще больше

- Объясните, как работают эти условные конструкции

```
if /l*o/.match("hello")  
  # are we here?  
end
```

```
if "hello" =~ /l*o/  
  # are we here?  
end
```

- Будет ли работать вот такое условие? Чем будет равна переменная md?

```
if md = "hello".match(/l*o/  
  # are we here?  
end
```

будет работать, т.к. md не nil  
md объект класса MatchData

```
if md = "hello" =~ /l*o/  
  # are we here?  
end
```

будет работать, т.к. md не nil  
md объект класса Fixnum, =2

см. [lesson.25/match\\_data/test\\_match\\_data\\_1](#)

# Получение последнего метча

- Даже если не захватывать MatchData в отдельную переменную, есть возможность получить результат последнего метча:

см. lesson.25/match\_data/test\_match\_data\_2

- после метча Руби устанавливает ряд **глобальных переменных**, которые доступны до следующей операции match, например, \$~, \$1, \$2 и др.

```
"hello, world" =~ /(l*o),\s(lw+)/  
puts $1  
puts $2
```

```
==> llo  
==> world
```

```
$~ – MatchData  
$1 – первая скобка  
$2 – вторая скобка
```

- Regexp.last\_match -- хранит экземпляр MatchData, результат последней операции {String,Regexp}#match.

Этот же объект хранится в переменной \$~

- Если последующий метч был неуспешным, автоматические глобальные переменные сбрасываются в nil.

см. lesson.25/match\_data/test\_match\_data\_6

## #чтоонетак!

- ✦ Вызов `match` внутри метода не изменяет внешнее (по отношению к методу) состояние глобальных переменных, устанавливаемых `match`.

см. `lesson.25/match_data/test_match_data_7`

- другие же глобальные переменные могут быть изменены внутри метода, так что вне метода будет видно новое значение.

# Задания

- ♦ Выясните, что содержится в `Regex.last_match` в скрипте

`lesson.25/match_data/test_match_data_3`

```
str = "Hello, World"  
str =~ /[[:punct:]]/ && str =~ /[[:upper:]]/
```

Результат какой из двух операций `String# =~` будет доступен через `Regex.last_match`?

Ответ: второй, будет замечена прописная H

- ♦ Выясните, что содержится в `Regex.last_match` в скрипте

`lesson.25/match_data/test_match_data_4`

```
str = "Hello, World"  
str =~ /[[:punct:]]/ || str =~ /[[:upper:]]/
```

Ответ: первой, будет замечен знак пунктуации

# Задания

- Исследуйте скрипт `lesson.25/match_data/test_match_data_5`. Как изменится результат, если убрать круглые скобки вокруг операций присваивания?

без скобок, результат будет равносильен

```
md1 = md2 = str.match(/[[[:upper:]]/)
```

как будто бы скобки были расставлены следующим образом

```
md1 = ( str.match(/[[[:punct:]]/)  
      && ( md2 = str.match(/[[[:upper:]]/) ) )
```

# Буквальное значение метасимволов

- Чтобы заметить метасимволы буквально, их необходимо экранировать при помощи обратной дроби или обрамляя каждый символ скобками:

метасимволы: . ? \* ( ) [ ] ^ - | \

- можно сразу записывать РВ с экранированными символами:

```
"5 a.m." =~ /a\.m\./  
"5 a.m." =~ /a[.]m[.] /
```

#=> 2

```
"5 aamm" =~ /a\.m\./  
"5 aamm" =~ /a[.]m[.] /
```

#=> nil

```
"5 a.m." =~ /a.m./  
"5 aamm" =~ /a.m./
```

#=> 2

- можно использовать специальный метод

Regexp.escape

```
str = '. ? * ( ) [ ] ^ - | \\  
puts str
```

```
puts Regexp.escape(str)
```

обратную дробь надо экранировать при вводе!

⇒ . ? \* ( ) [ ] ^ - | \

⇒ \. \? \\* \( \) \[ \] \^ \- \| \\



# Задания

- ♦ Какие символы будут экранированы методом Regexp.escape в строке

str = "he:.llo!"      #=> he:\.llo!

- ♦ Изучите скрипты:

- ♦ lesson.25/test\_safe\_match\_1

- ♦ lesson.25/test\_safe\_match\_2 – о строках в " и в "" и обратных дробях

```
re = /#{Regexp.escape(str)}/
```

- интерполяция внутри // возможна
- применять Regexp.escape необходимо к **строке**, а не к регулярному выражению

- ♦ lesson.25/find\_words\_with\_regexp/find\_words\_with\_regexp

```
queries = [ "Dec.", "U.S. Research Group" ]
```

```
str = queries.map do |query|  
  Regexp.escape(query)  
end.join('|')
```

#=> "Dec\.|U\.S\.\ Research\ Group"



## До и после

- MatchData позволяет получить доступ к фрагментам строки

```
str = "X_NN provides_VBZ a_AT flexible_JJ and_CC powerful_JJ Y_NN for_IN Z_NN"
```

```
md = str.match /\w+_JJ[RT]? \w+_CC \w+_JJ[RT]?/
```

```
> md.pre_match    #=> "X_NN provides_VBZ a_AT "
```

```
> md.post_match   #=> " Y_NN for_IN Z_NN"
```

- Объект класса MatchData ведет себя как массив, под индексом 0 находится замеченная подстрока:

```
> md[0]           #=> "flexible_JJ and_CC powerful_JJ"
```

- Синонимичные глобальные переменные:

```
$`  то же, что и MatchData#pre_match  
$'  то же, что и MatchData#post_match  
$&  то же, что и MatchData#[0]
```

- см. `lesson.25/test_finding_all_matches_with_post_match`

др. реализации этой задачи: `lesson.25/test_finding_all_matches`  
`lesson.25/test_finding_all_matches_with_scan`

# Accessing backreferences

- ♦ Как достать то, что было замечено скобками (backreference)

MatchData#captures	==> array
MatchData#to_a	==> array
MatchData#[]	

str = "X\_NN provides\_VBZ a\_AT flexible\_JJ and\_CC powerful\_JJ Y\_NN for\_IN Z\_NN"

```
md = str.match /((\w+)_JJ[RT]?) (\w+)_CC ((\w+)_JJ[RT]?) /
```

см. match\_data/test\_captures

- ♦ Глобальные переменные \$1, \$2 ... \$N также позволяют получить подстроки, замеченные отдельными скобками
- ♦ Вопрос: что содержит глобальная переменная \$0?

Ответ: имя скрипта, \$0 не имеет отношения к MatchData

## submatch offsets

- Следующие методы позволяют получить номера позиций в строке, где начинается/заканчивается подстрока, замеченная скобкой.

MatchData#begin(n)	==> Fixnum
MatchData#end(n)	==> Fixnum
MatchData#offset(n)	==> [Fixnum, Fixnum]

где n это номер скобки

и n=0 соответствует всему метчу

см. [lesson.25/match\\_data/test\\_offsets](#)

- Вопрос: укажите оффсеты (начала и конца) всей замеченной подстроки (на основании скрипта test\_offsets)

Ответ: [23, 53]

заменив на 0.upto(md.length-1)

==> parenthesis #0: 'flexible\_JJ and\_CC powerful\_JJ', starts at 23, ends at 53, offset [23, 53]

- Расскажите алгоритм, реализованный в скрипте

см. [lesson.25/test\\_finding\\_all\\_matches](#)

# Задание

➤ Реализуйте функциональность поиска всех метчей в строке в виде одного метода `all_matches`, который возвращает массив объектов `MatchData`. Поместите этот метод в модуль `Syn` (файл `syn.rb`), так чтобы его можно было использовать вот таким способом:

```
require 'syn' # или require 'syn_stub'  
...  
matches = Syn.all_matches(str, regexp)
```

начальный скрипт (с тестами): `lesson.25/syn_stub.rb`

ответ: `lesson.25/syn.rb`

● Чтобы выполнить тесты, необходимо запустить скрипт вот так:

```
> ruby syn.rb  
> ruby syn_stub.rb
```

`__FILE__` – имя файла, содержащего данный код.

`$0` – самый первый аргумент в командной строке, то есть, имя скрипта

```
if __FILE__ == $0  
  # ...  
end
```

# Памятка начинающего криптографа

- ♦ Глобальные переменные, которые устанавливает `Regex#match`

<code>\$~</code>	содержит последний установленный объект <code>MatchData</code> , = <code>Regex.last_match</code>
<code>\$&amp;</code>	заметченная строка целиком, = <code>MatchData#[0]</code>
<code>\$`</code>	подстрока левее заметченной, = <code>MatchData#pre_match</code>
<code>\$'</code>	подстрока правее заметченной, = <code>MatchData#post_match</code>
<code>\$1, \$2...</code>	подстрока, заметченная скобкой под указанным номером, = <code>MatchData#[1]</code> , <code>MatchData[2]</code> , ...
<code>\$+</code>	подстрока, заметченная последней скобкой, = <code>MatchData#[-1]</code>

# Именованные скобки

- Скобкам внутри РВ можно давать имена

(?<name>regexp)

(?<masha>\w+)

str = "Apocalypse starts **Feb. 30, 2014**, don't be late"

md = str.match /( ?<month>\w+[.]) ( ?<date>\d+), ( ?<year>\d+)/

- MatchData ведет себя не только как Array, но и как Hash

```
puts md[:month]      #=> "Feb."  
puts md[:date].to_i  #=> 30  
puts md[:year].to_i  #=> 2014
```

см. также [lesson.25/match\\_data/test\\_named\\_groups\\_1](#)

## Именованные скобки - 2

➤ Использовать именованные скобки удобно, чтобы избежать ошибок в счете скобок.

➤ В какую скобку попадает год в следующем РВ?

```
str = "Apocalypse starts Feb. 30, 2014, don't be late"
```

```
months = "((Jan|Feb|Mar|Apr|Jun|Jul|Aug|Sept?|Oct|Nov|Dec)[.]|May)"
```

```
md = str.match /({months})( \d+)?, (\d+)/
```

см. [lesson.25/match\\_data/test\\_named\\_groups\\_2](#)



# Non-capturing parenthesis

- ▶ Незахватывающие скобки (не попадают в captures)

`(?:regex)`

lesson.25/match\_data/test\_named\_groups\_2  
lesson.29/test\_noncapturing\_parens\_1

"the 10 apples"

`str.match(/(\w+)( \d+)? (\w+)/)` => `["the", " 10", "apples"]`

`str.match(/(\w+)(?: \d+)? (\w+)/)` => `["the", "apples"]`

- ▶ Задание: исследуйте состав captures, если `str = "the apples"`

"the apples"

`str.match(/(\w+)( \d+)? (\w+)/)` => `["the", nil, "apples"]`

`str.match(/(\w+)(?: \d+)? (\w+)/)` => `["the", "apples"]`



# Опции/модификаторы в РВ

➤ буквы m i x o

```
/re/i  
/re/im
```

/re/i Ignore case

/re/m Treat a newline as a character matched by .

/re/x Ignore whitespace and comments in the pattern

/re/o Perform #{} interpolation only once

➤ Назовите все известные способы заметить одну букву латинского алфавита (plain ascii) в любом регистре

/[A-Za-z]/

/[A-Z]/i

/[a-z]/i

/[[:alpha:]]/ или /\w/

/[[:upper:]]/i

/[[:lower:]]/i

## Опция /i

- ▶ Работает ли опция /i для символов из latin-1?

см. lesson.29/test\_diacritics



- ▶ Задание: на базе скрипта test\_diacritics исследуйте
  - ◆ включают ли символьные классы [:upper:] [:lower:] [:alpha:] символы с диакритиками?
  - ◆ работает ли опция /i для символьных классов

Ответ: да. кроме немецкой буквы ß, которая попадает только в класс lower

# Опции для отдельных скобок

- Опции `m` и `i` могут применяться к отдельным скобкам внутри одного РВ

```
/(?i:re1)re2/  
/(?im:re1)re2/
```

- Какие из фраз будут замечены следующим РВ?

`/(?i:the) apple/`

- the apple
- The apple
- THE apple
- the Apple

# Regex#to\_s

- Что же показывает Regex#to\_s?

```
puts /[A-Z]/i
```

#=> (?i-mx:[A-Z])

```
(?i-mx: ... )
```

- Включение и выключение опций:

```
(?on-off: ... )
```

- Какое из РВ будет метчить без учета регистра?

```
(?m-ix: ... )
```

```
(?mi-x: ... )
```



# Комментарии в РВ

➤ Два способа сделать РВ понятнее

➤ Способ 1

(?#comment)

Например, РВ для описания времени в формате ЧЧ:ММ

```
/(?#hours)([01][0-9]|2[0-4]):(?#minutes)([0-5][0-9])/
```

см. [lesson.29/test\\_comments\\_1](#)

## Комментарии в РВ - 2

- Способ 2 – опция /x позволяет
  - ◆ записывать РВ на нескольких строках
  - ◆ вставлять комментарии в конце строк

РВ для описания времени в формате ЧЧ:ММ:СС

```
/^([01][0-9]|2[0-4])  # hours
:([0-5][0-9])         # minutes
:([0-5][0-9])         # seconds
$/x
```

см. lesson.29/test\_comments\_2

- NB: пробельные символы и комментарии (начинающиеся с #) игнорируются
- Задание: измените РВ (в режиме /x) для времени так, чтобы допускались пробелы вокруг двоеточия, что есть, чтобы метчились

10:45  
10 : 45

начальный скрипт: lesson.29/test\_comments\_3\_stub  
ответ: lesson.29/test\_comments\_3

# Опция /m

- Метасимвол . метчит любой символ кроме \n

```
str = "the apples  
and pears"
```

С опцией /m метасимвол . будет включать \n

```
str.match(/.+/)
```

метчит the apples

```
str.match(/.+/m)
```

метчит все строку

см lesson.29/test\_dot\_with\_m

# Якоря ^ \$ vs. \A \Z

## Из документации ri Regexp

^	Matches beginning of <b>line</b>
\$	Matches end of <b>line</b>
\A	Matches beginning of <b>string</b> .
\Z	Matches end of <b>string</b> . If string ends with a newline, it matches just before newline

## Сколько здесь lines и сколько здесь strings? попробуйте следующие случаи

```
str = "spring  
has  
come"
```

str.match(/^./)	#=> s
str.match(/^./, 2)	#=> h
str.match(/^./, 8)	#=> c

str.match(/\A./)	#=> s
str.match(/\A./, 2)	#=> nil

Ответ: здесь 3 lines и одна string. line это то, что разделено символом \n



# Якоря \$ vs. \Z

- Что заметчат следующие РВ

```
str = "spring  
has  
come"
```

<code>str.match(/.\$/)</code>	<code>#=&gt; g</code>
<code>str.match(/.\$/,6)</code>	<code>#=&gt; s</code>
<code>str.match(/.\Z/)</code>	<code>#=&gt; e</code>

- Анатомия строки с точки зрения Руби

```
\A^spring$\n  
^has$\n  
^come$\Z\n
```

# Задание

- ♦ Исследуйте, как влияет на якоря опция /m

```
str = "spring  
has  
come"
```

```
str.match(/^./)    #=> spring  
str.match(/A./)    #=> spring  
str.match(/^./m)   #=> все от начала до конца  
str.match(/A./m)   #=> все от начала до конца
```

```
str.match(/^./, 2)  #=> has  
str.match(/^./m, 2) #=> nil  
str.match(/A./, 2)  #=> nil  
str.match(/A./m, 2) #=> nil
```

Предположение: в режиме /m якорь ^ ведет себя как \A

## (продолжение)

```
str = "spring  
has  
come"
```

```
str.match(/.+$/ )      #=> spring  
str.match(/.+$/m)      #=> вся строка  
str.match(/.+$/, 6)     #=> has  
str.match(/.+$/m, 6)    #=> \nhas\ncome
```

```
str.match(/.+\\Z/)      #=> come  
str.match(/.+\\Z/m)     #=> вся строка  
str.match(/.+\\Z/, 6)   #=> come  
str.match(/.+\\Z/m, 6)  #> \nhas\ncome
```

# Regex.union

- объединяет данные шаблоны в одну *дизъюнкцию*

```
Regex.union(ptrn1, ptrn2,...)
Regex.union([ptrn1, ptrn2,...])
```

#=> /ptrn1|ptrn2/

- Шаблоны могут быть заданы в форме объектов String или Regex
  - если String, то к ней применяется `Regex.escape`
  - если `Regex`, то используется как есть

```
Regex.union "d.gs", /d.cks?/i
```

#=> /d\.gs|(?i-mx:d.cks?)/

см. [lesson.31/regexp/test\\_regexp\\_union](#)

- Какие из приведенных строк заметит это РВ?

❖ "dogs"

❖ superduck

❖ "d.gs"

❖ "d\.gs"

❖ superdUck

❖ "hotd.gs"

❖ 'd\.gs'

❖ superdUCk

Потому что опция i ?

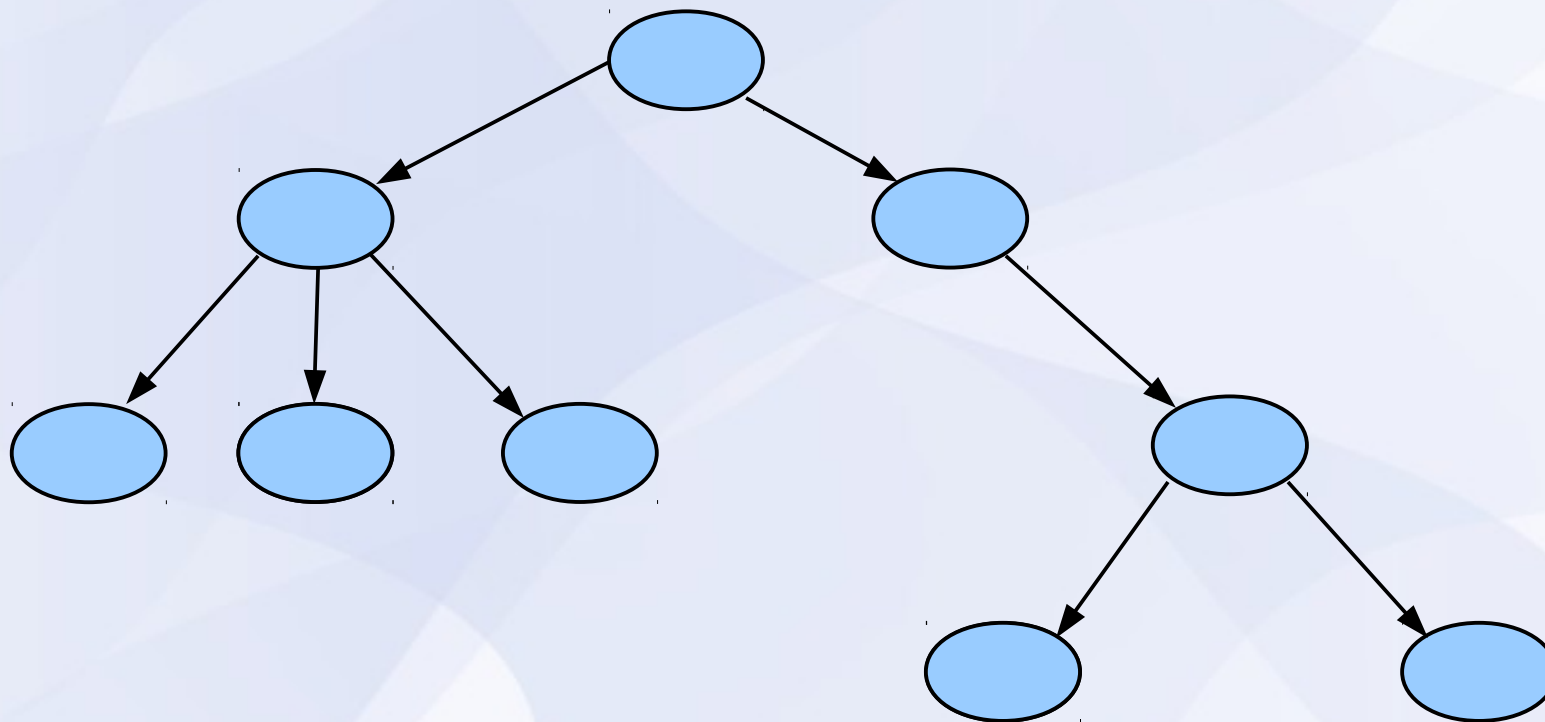
Нет, потому что ./ !

# Рекурсия

- \* чтобы понять рекурсию, нужно понять рекурсию
- \* разделяй и властвуй!

# Рекурсивные структуры данных

- ♦ Дерево: родитель -> дочерние узлы, дочерние узлы в свою очередь являются родительскими для других узлов



# В математике

♦ Функция может быть задана рекурсивно, при помощи так называемой *рекуррентной формулы*.

- ♦ функция задана для некоторого начального значения  $a$  (*base case*)
- ♦ если функция задана для некоторого значения  $k$ , большего  $a$ , то она задана также для значения  $k+1$  (*inductive step*). Проще говоря, каждый элемент последовательности задан через предыдущий/ие элемент/ы этой последовательности.

$$f(n) = a^n \quad \text{для } n \geq 0$$

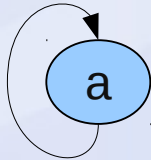
$$f(0) = 1$$

$$f(k+1) = a * f(k)$$

$k \rightarrow n$	$f(k) = a^k$ напр. $2^5$	результат на этом шаге
0	$f(0) = 1$	1
1	$f(1) = f(0+1) = a * f(0)$	$2 * 1 = 2$
2	$f(2) = f(1+1) = a * f(1)$	$2 * 2 = 4$
3	$f(3) = f(2+1) = a * f(2)$	$2 * 4 = 8$
4	$f(4) = f(3+1) = a * f(3)$	$2 * 8 = 16$
5	$f(5) = f(4+1) = a * f(4)$	$2 * 16 = 32$

# В программировании

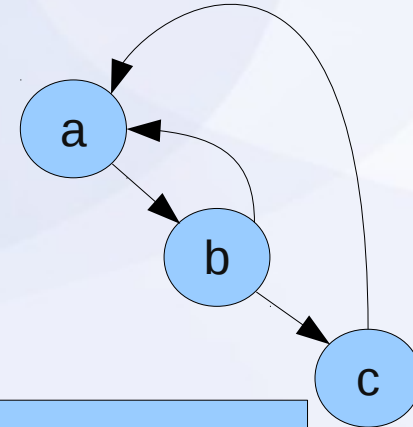
➤ Рекурсивная процедура - процедура, *прямо или косвенно* вызывающая саму себя.



```
def a(arg)
  ...
  a(modified_arg)
  ...
end
```

*Разделяй и властвуй:*

задача разбивается на **более маленькие задачи**, которые могут быть решены **тем же самым способом** (вызовом того же самого метода). Общее решение **выводится из решений** частей.



```
def a(arg)
  ...
  b(modified_arg)
end

def b(arg)
  ...
  a(modified_arg)
end
```



# Forward/Backward recursion

## ♦ прямая рекурсия:

см. recursion/power\_forward\_rec

- ♦ начинаем с базового случая
- ♦ решив более простой случай задачи длины N, переходим к решению задачи длины N+1
- ♦ останавливаемся, когда достигнуто нужное N

♦ прямая рекурсия выглядит как замена цикла: есть подобие итерационной переменной (k) с начальным и конечным значением

## ♦ обратная рекурсия:

- ♦ начинаем с решения задачи длины N
- ♦ и принимаем, что задача для N-1 уже решена
- ♦ когда N-1 приходит к базовому случаю, останавливаемся

cool!

$$f(n) = a^n \quad \text{для } n \geq 0 \quad \left\{ \begin{array}{l} f(0) = 1 \\ f(k) = a * f(k-1) \end{array} \right.$$

# Трассировка рекурсии

→ см. recursion/power\_rec

$$f(n) = a^n \quad \text{для } n \geq 0 \quad \left\{ \begin{array}{l} f(0) = 1 \\ f(k) = a * f(k-1) \end{array} \right.$$

```
def power a, n
  if n == 0
    return 1
  else
    return a * power(a, n-1)
  end
end
puts power(2,5) #=> 32
```

“ждущее” значение

№ шага	аргументы	“ждущее” значение	результат вложенного вызова метода	результат шага
1	power(2,5)	2	16	32
2	power(2,4)	2	8	16
3	power(2,3)	2	4	8
4	power(2,2)	2	2	4
5	power(2,1)	2	1	2
6	power(2,0)	-	-	1

# Base case(s)

- ♦ base case (также граничное условие) – простейшая форма подзадачи, для решения которой не требуется рекурсивный вызов
- ♦ Подходит ли для задачи вычисления  $a^n$  для  $n \geq 0$  такой base case?

```
if n == 1  
  return a  
end
```

не подходит. не рассмотрен случай  $n = 0$

```
if n == 0  
  return 1  
elsif n == 1  
  return a  
end
```

подходит.

мораль: базовых случаев может быть много

- ♦ Для строк/массивов базовым случаем часто является строка/массив длины ноль

# Несколько/разные рекурсивные вызовы

- ♦ [http://en.wikipedia.org/wiki/Collatz\\_conjecture](http://en.wikipedia.org/wiki/Collatz_conjecture)
- ♦ для любого положительного числа
  - ♦ If the number is even, divide it by two.
  - ♦ If the number is odd, triple it and add one.
- This process will eventually reach the number 1, regardless of which positive integer is chosen initially.

```
def collatz n
  if n == 1
    1
  elsif n.even?
    collatz n/2
  else
    collatz 3*n+1
  end
end
```

Мораль: рекурсивных вызовов может быть несколько и они могут быть разными.

см. [lesson.31/recursion/collatz\\_conjecture](#)

# Задания

- ♦ Разработайте рекурсивную реализацию переворота строки. Заполните таблицу трассировки для примера:

reverse("bamboo") #=> oobmab

- ♦ Base case? строку длины 0 не надо перворачивать
- ♦ Индукция? *отделить* первый символ и *присоединить* его в конец *перевернутого* остатка («остаток» – строка без первого символа)

Ответ: recursion/reverse\_string

reverse("bamboo")	b	oobma	oobmab
reverse("amboo")	a	oobm	oobma
reverse("mboo")	m	oob	oobm
reverse("boo")	b	oo	oob
reverse("oo")	o	o	oo
reverse("o")	o	""	o
reverse("")	-	-	""

# Задания

- Реализуйте метод, вычисляющий факториал заданного неотрицательного числа

$$n! = 1 * 2 * 3 \dots n$$

$$5! = 1 * 2 * 3 * 4 * 5 = 120$$

$$\left\{ \begin{array}{l} \text{факториал}(0) = 1 \\ \text{факториал}(k+1) = (k+1) * \text{факториал}(k) \end{array} \right.$$

Ответ: `lesson.31/recursion/factorial`

# Задания

- Разработайте метод, (рекурсивно) вычисляющий сумму всех элементов массива.

Начальный скрипт: `lesson.31/recursion/array_sum_stub`

Ответ: `lesson.31/recursion/array_sum`

- Разработайте метод, (рекурсивно) вычисляющий сумму N первых элементов массива (N передается как аргумент) .

```
nums = [1,10,1,4,5]
```

```
array_sum(nums, 4)
```

```
array_sum(nums, 2)
```

```
array_sum(nums, 10)
```

Ответ: `lesson.31/recursion/array_sum_n_first`

```
#=> 16
```

```
#=> 11
```

```
#=> 21
```

- Сравните данные реализации `array_sum` и `array_sum_n_first`. Какое из решений лучше использует память?
- Переделайте `array_sum` так, чтобы не создавались лишние экземпляры массивов для каждого рекурсивного вызова.

## В следующих сериях

- хвостовая рекурсия и что ее роднит с земноводными
- фибоначчи и популяционная биология на примере кроликов
- memoization без r
- взаимная рекурсия
- и снова палиндромы



# Заголовок