

# EPI: Interfejs Graficzny 2011/2012

## Wykład nr 2

### Wbudowane typy danych Rubiego

Aleksander Pohl

7 listopada 2011

# Plan prezentacji

Łańcuchy

Liczby

Symbole i zakresy

W. regularne

Kontenery

Pliki

Materiały

## Łańcuchy znaków (Ruby 1.9)

- ▶ sekwencje znaków (a nie bajtów)
- ▶ wsparcie dla Unicode oraz innych kodowań
- ▶ ograniczane za pomocą ' lub "

- ▶ cytowanie apostrofów:

```
'test pojedynczego cudzysłowu: \'
```

```
"test podwójnego cudzysłowu: \"
```

- ▶ tylko łańcuchy ograniczone podwójnymi cudzysłowami mogą być „interpolowane”:

```
value = 'EPI'
```

```
"Witaj #{value}!"
```

```
'Witaj #{value}!'
```

# Łańcuchy wielowierszowe

zaczynają się od pary znaków << i ciągu znaków, który ma symbolizować koniec łańcucha:

```
str = <<END_OF_STRING
      Siała baba mak
      Nie wiedziała jak
      Dziadek wiedział
      Nie powiedział
      A to było tak...
END_OF_STRING
```

# Operacje na łańcuchach

„Mnożenie” i konkatelowanie:

```
"Witaj "+"EPI"      #=> "Witaj EPI"
"EPI" *3            #=> "EPI EPI EPI "
```

```
greeting = "Witaj"
greeting << " EPI"
greeting << "!!!"   #=> "Witaj EPI!!!"
```

Badanie zawartości łańcucha:

```
"Test".empty?      #=> false

"świat".length     #=> 5, zarówno dla kodowani ISO jak i UTF-8
"świat".size       #=> 5, to samo co length

"paczka".index('a') #=> 1
"paczka".rindex('l') #=> 5

"świat"[0..3]      #=> "świa"
"świat"[-3..-1]   #=> "iat"
```

# Operacje na łańcuchach

## Manipulowanie łańcuchami:

```

"TEST".downcase      #=> "test"
"ŚWIAT".downcase    #=> "Świat" !!!
require 'string_case_pl'
"ŚWIAT".downcase    #=> "świat"

"223".to_i           #=> 223
"223.5".to_i         #=> 223
"223.5".to_f        #=> 223.5
"FF".to_i(16)       #=> 255
"101010".to_i(2)    #=> 42

"zbyt wiele   spacji   ".squeeze(" ") #=> "zbyt wiele spacji "
"   początkowe i końcowe spacje   ".strip #=> "początkowe i końcowe spacje"
"Cześć! Jak się masz?".split           #=> ["Cześć!", "Jak", "się", "masz?"]
"Cześć! Jak się masz?".split(/([!\\?] ?)/) #=> ["Cześć!", "Jak się masz"]
"hello \n".chomp      #=> "hello "
"hello...".sub(/\\.\/, '!') #=> "hello!.."
"hello...".gsub(/\\.\/, '!') #=> "hello!!!"

```

# Operacje na łańcuchach

Konwencja nazewnicza: operacje modyfikujące i niemodyfikujące

- ▶ `downcase` - zwraca nowy łańcuch
- ▶ `downcase!` - modyfikuje oryginalny łańcuch

```
first_name = "Janek"
first_name.downcase      #=> "janek"
puts first_name          #=> Janek
first_name.downcase!    #=> "janek"
puts first_name          #=> janek
first_name.downcase!    #=> nil !
```

Łączenie operacji:

```
"POWITANIE    ZE    SPACJJAMI".squeeze.strip.downcase
#=> "powitanie ze spacjami"
"powitanie ze spacjami".squeeze!.strip!.downcase!
# NoMethodError: undefined method 'strip!' for nil:NilClass
```

# Liczby

- ▶ liczby całkowite – obiekty klasy Fixnum lub Bignum
- ▶ liczby zmiennopozycyjne – obiekty klasy Float

zamiana typu jest automatyczna:

```
a = 5           #=> 5
a.class        #=> Fixnum
a = a + 1.0    #=> 6.0
a.class        #=> Float
```

ale może prowadzić do niespodzianek:

```
a = 5
a / 2           #=> 2
a / 2.0        #=> 2.5
```

jawna konwersja typów:

```
7.5.to_i       #=> 7
8.to_f         #=> 8.0
8.5.to_s       #=> "8.5"
8.to_s(2)      #=> "1000"
```

# Podstawowe operacje na liczbach

Przypisanie równoległe (nie tylko liczby):

`a = 2`

`a,b = 3,4`

Operacje arytmetyczne:

|                           |                    |
|---------------------------|--------------------|
| <code>+</code>            | dodawanie          |
| <code>-</code>            | odejmowanie        |
| <code>*</code>            | mnożenie           |
| <code>/</code>            | dzielenie          |
| <code>%</code>            | dzielenie modulo   |
| <code>**</code>           | potęgowanie        |
| <code>Math::log()</code>  | logarytm naturalny |
| <code>Math::sqrt()</code> | pierwiastek        |

# Podstawowe operacje na liczbach

Obiektowa interpretacja operacji arytmetycznych:

```
2 + 3 * 7 #=> 2.+(3.*(7))
```

Operatory zachowują swoje naturalne priorytety (np. mnożenie wykonywane jest przed dodawaniem).

Operatory porównania: `<`, `<=`, `==`, `>=`, `>`

`<=>` komparator:

```
2 <=> 2 #=> 0
```

```
2 <=> 3 #=> -1
```

```
2 <=> 0 #=> 1
```

# Liczby jako obiekty

```

0.zero?           #=> true
0.nonzero?       #=> false
10.between?(12,15) #=> false
2.5.integer?     #=> false

b = 5.5
b.round          #=> 6
c = -7
c.abs            #=> 7
7.succ           #=> 8

3.times{|i| puts "Witaj EPI po raz #{i+1}!"}
# Witaj EPI po raz 1!
# Witaj EPI po raz 2!
# Witaj EPI po raz 3!

require 'active_support/time'
3.days.ago       #=> 2011-10-08 10:06:54 +0200

```

# Symbole

Symbol reprezentuje w Rubim pewną nazwę.  
Jest tworzony automatycznie poprzez użycie :dwukropka na początku nazwy:

```
:ruby
```

Tylko jeden obiekt klasy Symbol o danej nazwie jest tworzony w ciągu całego wykonania programu:

```
f1,f2 = :ruby,:ruby  
f1.object_id  
f2.object_id #ten sam obiekt
```

```
f1,f2 = "ruby","ruby"  
f1.object_id  
f2.object_id #dwa różne obiekty
```

# Zakresy

```
1..10 (zawiera 10)
1...10 (nie zawiera 10)
a = 1..10
a.min #=> 1
a.max #=> 10
a.include?(10) #=> true
```

jako sekwencje:

```
(1..10).to_a #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

jako interwały:

```
(1..10) === 5 #=> true
(1..10) === 3.1415 #=> true
```

# Wyrażenia regularne

- ▶ obiekty klasy Regexp
- ▶ zapisywane z wykorzystaniem ukośników, np. `/he1.*/`
- ▶ definiują wzorce dopasowywane do łańcuchów
- ▶ wszystkie znaki dopasowywane są do siebie samych, z wyjątkiem znaków specjalnych (metaznaków): `.`, `|`, `(`, `)`, `[`, `{`, `+`, `\`, `^`, `$`, `*`, `?`
- ▶ aby dopasować znak specjalny, należy poprzedzić go odwrotnym ukośnikiem `\`
- ▶ `=~` operator dopasowania

# Kotwice

- ▶ `^` dopasowuje początek linii
- ▶ `\A` dopasowuje początek łańcucha
- ▶ `$` dopasowuje koniec linii
- ▶ `\Z` dopasowuje koniec łańcucha

```
/hel/           # dopasowuje "hel" w dowolnym miejscu łańcucha
```

```
/hel/ =~ "hello"  #=> 0 - indeks początku dopasowania
```

```
/hel/ =~ "goodbye" #=> nil - brak dopasowania
```

```
/^hel/         # dopasuj tylko na początku
```

```
/hel$/        # dopasuj tylko na końcu
```

# Powtórzenia i alternatywa

Powtórzenia:

jeśli  $r$  jest aktualnym znakiem, to:

$r^*$  dopasowuje zero lub więcej wystąpień  $r$

$r^+$  dopasowuje jedno lub więcej wystąpień  $r$

$r^?$  dopasowuje zero lub jedno wystąpienie  $r$

$r\{m,n\}$  dopasowuje co najmniej „ $m$ ” i co najwyżej „ $n$ ” wystąpień  $r$

$r\{m,\}$  dopasowuje co najmniej „ $m$ ” wystąpień  $r$

"aaaabc"  $\approx$  `/a*/`  $\#=>$  dopasuje "aaaa"

"aaaabc"  $\approx$  `/a*b?/`  $\#=>$  dopasuje "aaaab"

"aaaac"  $\approx$  `/a*b?/`  $\#=>$  dopasuje "aaaa"

"abaac"  $\approx$  `/a{2}/`  $\#=>$  dopasuje "aa"

Alternatywa:

$a|b$  dopasowuje jedną z alternatyw

"witaj"  $\approx$  `/witaj|żegnaj/`  $\#=>$  dopasuje "witaj"

# Operacje na wyrażeniach regularnych

Klasy znaków:

[] – dopasowuje dokładnie jeden ze znaków wymienionych w nawiasie

Skróty klas znaków

|        |               |   |
|--------|---------------|---|
| .      |               | dowolny znak (oprócz końca linii)         |
| \d     | [0-9]         | cyfry                                     |
| \D     | [^0-9]        | nie-cyfry                                 |
| \s     | [\s\t\r\n\f]  | białe spacje                              |
| \S     | [^\s\t\r\n\f] | nie-białe spacje                          |
| \w     | [A-Za-z0-9_]  | znaki alfanumeryczne (ale nie a, e, itp.) |
| \W     | [^A-Za-z0-9_] | znaki nie-alfanumeryczne                  |
| \p{L}  | [[:letter:]]  | litery (np. A, B, c, a)                   |
| \p{Lu} | [[:upper:]]   | wielkie litery (np. A, B, C, A)           |

# Przykłady wyrażeń

```

/hel.*/           # dopasowuje "hello"
/hello!?!*/      # dopasowuje "hello" i "hello!"
/hello!+*/       # dopasowuje "hello!!!!", ale nie "hello"
/hello[!\.]/     # dopasowuje "hello!" i "hello."

/^[A-Z][a-z]+$/  # dopasowuje "Warszawa", ale nie "Łódź"
/^\p{Lu}\p{Ll}+$/ # dopasowuje "Warszawa" oraz "Łódź"
/^\ala/          # dopasowuje "ala" oraz "kot\nala"
/^\Aala/         # dopasowuje "ala", ale nie "kot\nala"

/^\d{2}-\d{3}$/  # dopasowuje "33-310", "44-100", etc.
/Ala i kot|Jola/ # dopasowuje "Ala i kot" i "Jola"
                 # dla "Ala i Jola" dopasowuje tylko "Jola"
/^\Ala i kot|Jola$/ # dopasowuje "Ala i kot" na początku linii
                 # oraz "Jola" na końcu linii

```

# Grupowanie

() – wszystko wewnątrz nawiasów traktowane jest jako pojedyncze wyrażenie

```
/hello (John|James)!/
```

Przechwytywanie podgrup:

```
match_data = /([a-zA-Z]+) ([a-zA-Z]+)/.match("123Jan Kowalski456")
```

```
#=> #<MatchData:0xb7aa6b8c>
```

```
match_data[0]           #=> "Jan Kowalski", inaczej $0
```

```
match_data[1]           #=> "Jan", inaczej $1
```

```
match_data[2]           #=> "Kowalski", inaczej $2
```

```
match_data.pre_match    #=> "123", inaczej $'
```

```
match_data.post_match   #=> "456", inaczej $'
```

```
"123Jan Kowalski456".sub(/([a-zA-Z]+) ([a-zA-Z]+)/, "\\2 \\1")
```

```
#=> "123Kowalski Jan456"
```

# Tablice

```

arr = ["Fred",1,3.14]
arr[0]      #=> "Fred"
arr[1]      #=> 1
arr[-1]     #=> 3.14
arr[-2]     #=> 1
arr[0..1]   #=> ["Fred",1]
arr[-2..-1] #=> [1, 3.14]

arr[0] = "Wilma"
arr        #=> ["Wilma",1,3.14]
arr[0..1] = ["Fred",10]
arr        #=> ["Fred",10,3.14]

Array.new      #=> []
Array.new(3)   #=> [nil, nil, nil]
Array.new(3,"a") #=> ["a", "a", "a"]

arr = %w(fred wilma barney betty the\ flintstones)
#=> ["fred", "wilma", "barney", "betty", "the flintstones"]

```

# Operacje na tablicach

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

```
b = [3,4]
```

```
a + b           #=> [1,2,3,3,4]
```

```
a - b           #=> [1,2]
```

```
a | b           #=> [1,2,3,4]
```

```
a & b           #=> [3]
```

```
a = [2,3]
```

```
a << 2           #=> [2,3,2]
```

```
a.push(5)       #=> [2,3,2,5]
```

```
a.unshift(1)    #=> [1,2,3,2,5]
```

```
a.pop           #=> 5
```

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

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

```
a.combination(2).to_a
```

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

```
a.permutation(2).to_a
```

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

# Operacje na tablicach

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

```
a.length      #=> 4
```

```
a.empty?      #=> false
```

```
a.reverse     #=> [4, 3, 2, 1]
```

```
a.first       #=> 1
```

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

```
a.shift       #=> 1
```

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

```
a.last        #=> 4
```

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

```
a.pop         #=> 4
```

```
a             #=> [2, 3]
```

# Operacje na tablicach

```
a = ["a", nil, "b", "b", nil, "c"]

a.compact      #=> ["a", "b", "b", "c"]
a              #=> ["a", nil, "b", "b", nil, "c"]
a.compact!    #=> ["a", "b", "b", "c"]
a              #=> ["a", "b", "b", "c"]
a.uniq!       #=> ["a", "b", "c"]

a.join(", ")   #=> "a, b, c"
a * ", "      #=> "a, b, c"

a.delete("a")  #=> "a"
a              #=> ["b", "c"]
a.delete_at(1) #=> "c"
a              #=> ["b"]

a == ["b"]     #=> true
```

# Tablice asocjacyjne

```
h = {"name" => "Fred", "surname" => "Flintstone"}
h["name"]           #=> "Fred"

h["name"] = "Wilma"
#=> {"name" => "Wilma", "surname" => "Flintstone"}
h[:name] = "Wilma"
#=> {"name" => "Wilma", "surname" => "Flintstone", :name => "Wilma"}

h1 = Hash.new           # to samo co {}
h1[:foo]               #=> nil
h2 = Hash.new(0)
h2[:foo]               #=> 0
```

# Tablice asocjacyjne

```
h = {:foo => "bar", :bar => "baz"}
```

```
h.empty?           #=> false
```

```
h.size            #=> 2
```

```
h.length          #=> 2
```

```
h.include?(:foo)  #=> true
```

```
h.has_key?(:foo)  #=> true #synonim include?
```

```
h.has_value?("bar") #=> true
```

```
h.index("bar")    #=> :foo
```

```
h.keys            #=> [:bar, :foo]
```

```
h.values          #=> ["baz", "bar"]
```

```
h.delete(:foo)    #=> "bar"
```

```
h.clear           #=> {}
```

# Operacje na plikach

```
file = File.open("plik.txt")
file.each do |line|
  puts line
end
file.close

File.open("plik.txt","r:iso-8859-2") do |file|
  file.each do |line|
    puts line
  end
end

File.open("plik.txt","w:utf-8") do |file|
  10.times do |index|
    file.puts "Zażółć gęślą jaźń"
  end
end
```

# Operacje na plikach i katalogach

```
Dir.glob("*.txt").each do |file_name|
  File.open(file_name, "w") do |file|
    if file.file?
      file.chmod(0600)
    end
  end
end

`ls *.txt`.split("\n").each do |file_name|
  File.open(file_name, "w") do |file|
    if file.file?
      file.chmod(0600)
    end
  end
end
```

# Materiały

- ▶ Dokumentacja typów podstawowych  
<http://railsapi.com>
- ▶ Typy standardowe języka  
<http://www.apohllo.pl/dydaktyka/ruby/intro/typy-danych>

# Pytania

PYTANIA?