Programowanie w Ruby

Wykład 2

Marcin Młotkowski

15 października 2018

Plan wykładu

Liczby całkowite

Fixnum

Zakres: $[-2^{30} \ldots 2^{30} - 1]$
Przykłady: 119, 0xFFFF, 0b1101001
Konwencja: 8_388_608

Bignum

Dowolnie wielkie liczby
Automatyczna konwersja, jeśli wynik działania lub stała przekracza Fixnum

Operacje na typie Fixnum

wyrażenie	wynik
x <=> y	porównuje liczby, zwraca -1, 0, 1
123[2]	zwraca 2 bit (licząc od 0)
5.size	liczba bitów
5.to_f	przekształca na liczbę klasy Float
3.to_s	przekształcenie do String
15.to_s(2)	przekształcenie do String, ale przy podstawie 2
x.zero?	sprawdzenie, czy liczba jest równa zero

Liczby typu Float

- Zakres: [Float::MIN..FLOAT::MAX]
- Standardowe operatory, oraz operator <=>
- 3.14.to_s
- w.finite?

Ciekawostka

$ puts (−1.0/0.0)

nie zwraca błędu

Ciekawostka

$ puts (−1.0/0.0)

nie zwraca błędu

tylko *-Infinity*

Wyrażenia logiczne

- Fałsz: nil oraz false
- prawda: wszystko inne co nie jest fałszem;
- Operatory: and, &&, or, ||, not, !
- ==, defined?

Wyrażenia logiczne

- Fałsz: nil oraz false
- prawda: wszystko inne co nie jest fałszem;
- Operatory: and, &&, or, ||, not, !
- ==, defined?

Wartość wyrażenia logicznego

Wartością wyrażenia logicznego, gdy jest prawdziwe, jest ostatni operand.

Wyrażenia logiczne

- Fałsz: nil oraz false
- prawda: wszystko inne co nie jest fałszem;
- Operatory: and, &&, or, ||, not, !
- ==, defined?

Wartość wyrażenia logicznego

Wartością wyrażenia logicznego, gdy jest prawdziwe, jest ostatni operand.

Przykłady

'nie' and "tak"	'tak'
false and 99	false

Operatory logiczne

Kolejność obliczania wyrażeń logicznych (1)

x1 or x2 and x3:
(x1 or x2) and x3

Kolejność obliczania wyrażeń logicznych (2)

x1 || x2 && x3:
x1 || (x2 && x3)

Plan wykładu

Napisy (typ: String)

Przykłady:

'string', "string"

'alfabet Morse\'a'

"to jest \"cytat\""

Notacja Q

%q/Ciągłe 'cytowanie' jest 'bardzo' żmudne/

%Q!Ale "w" Ruby'm "bardzo" łatwe!

here documents

```
napis = <<END_OF_STRING
Ogary poszły w las.

Echo ich grania słabło coraz bardziej, aż wreszcie utonęło w
milczeniu leśnym.
END_OF_STRING
```

Tworzenie napisów

```
"Ho! " * 3   nowy napis "Ho! Ho! Ho! "
"A " << "kuku!" modyfikuje napis do postaci "A kuku!"
```

Wyrażenia wewnątrz napisów

```
"2+2=#{2 + 2}"

"Hip hip #{'Hura!' * 3}"

pi = 3.1415
puts "pi = #{pi}"
```

Operacje na napisach

```
irb(main):001:0> 'abc' + 'def'
=> "abcdef"
```

Operacje na napisach

```
irb(main):001:0> 'abc' + 'def'
=> "abcdef"

irb(main):001:0> x = 'abc'
=> "abc"
irb(main):002:0> x << 'def'
=> "abcdef"
```

Odwołania do elementów

wyrażenie	wynik
'abcdef'[3]	100
'abcdef'[3,1]	"d"
'abcdef'[3..4]	"de"
'abcdef'[-3..-1]	"def"
'abcdef'[/(d.f)/]	"def"

Modyfikacja napisów

Operacja	wartość zmiennej x
x = 'skówka'	x = 'skówka'
x[2] = 'u'	x = 'skuwka'
x[/w/] = 'f'	x = 'skufka'
x[0..1]= 'zas'	x = 'zasufka'

Modyfikowanie a tworzenie

Modyfikowany jest napis

operacje modyfikowania

[ ]=, <<

Napis nie jest modyfikowany, za to tworzony jest nowy, który jest wynikiem

operacje tworzenia

+, *

Pary metod

Zwraca kopię	modyfikuje napis	opis
"aaa".capitalize	"aaa".capitalize!	zamienia litery na wielkie
"str\n".chomp	"str\n".chomp!	usuwa białe znaki z końców napisu

Formatowanie tekstu

```
"Pi = %.2f" % 3.1415

"Imię %s nazw %s" % ['Jan', 'Kowalski']
```

Porównywanie napisów

eql?, ==, ===, equal?

```
str1 = "ala ma kota"
str2 = "ala ma kota"

str1 == str2
str1 === str2
str1.eql?(str2)
```

Inne operacje na stringach

Całe mnóstwo ;-)

Inne operacje na stringach

Całe mnóstwo ;-)

http://ruby-doc.org/core/
String

Dygresja: dokumentacja wbudowana RDoc

```
$ ri String
$ ri String.chomp
$ ri chomp
```

Znaki narodowe

Z dokumentacji klasy String

String to ciąg bajtów zwykle reprezentujących znaki.

Znaki narodowe

Z dokumentacji klasy String

String to ciąg bajtów zwykle reprezentujących znaki.

Konsekwencje

- metody capitalize, upcase, length etc. działają tylko dla ASCII;
- puts i print drukują poprawnie UTF-8

Znaki narodowe

Z dokumentacji klasy String

String to ciąg bajtów zwykle reprezentujących znaki.

Konsekwencje

- metody capitalize, upcase, length etc. działają tylko dla ASCII;
- puts i print drukują poprawnie UTF-8

Korzystanie z utf-8

$KCODE = "UTF–8"

Przykłady tablic

Tworzenie tablic

```
[1, 'dwa', 3.0]

%w{ poniedziałek wtorek środa }
                    daje: ["poniedziałek", "wtorek", "środa"]
```

Przykłady tablic

Tworzenie tablic

```
[1, 'dwa', 3.0]

%w{ poniedziałek wtorek środa }
```

daje: ["poniedziałek", "wtorek", "środa"]

Parę uwag o tablicach

- indeksowanie jest od zera
- dołączanie elementów na koniec
  [1, 2, 3] << 4 << 5
- Usuwanie ostatniego elementu
  arr.delete(arr.length - 1)

Operacje na tablicach

Odwołania do elementów

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1,3]
daje ['jeden', 'dwa', 'trzy']

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1..3]
daje ['jeden', 'dwa', 'trzy']

Operacje na tablicach

Odwołania do elementów

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1,3]
daje ['jeden', 'dwa', 'trzy']

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1..3]
daje ['jeden', 'dwa', 'trzy']

Modyfikacje

y = [1,1,2,3,5,8]
y[3..5] = [1,1,2]
daje [1, 1, 2, 1, 1, 2]

Operacje na tablicach

Odwołania do elementów

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1,3]
daje ['jeden', 'dwa', 'trzy']

['zero', 'jeden', 'dwa', 'trzy', 'cztery'][1..3]
daje ['jeden', 'dwa', 'trzy']

Modyfikacje

y = [1,1,2,3,5,8]
y[3..5] = [1,1,2]
daje [1, 1, 2, 1, 1, 2]

Więcej operacji na tablicach

$ ri Array

Przykłady przedziałów

Przedziały

```
1..7
0...256
'a'..'z'
```

Konwersja przedziału na tablicę

```
(1..5).to_a ==> [1,2,3,4,5]
('aac'..'aaf').to_a
  ==> ["aac", "aad", "aae", "aaf"]
```

Zastosowanie przedziałów

```
epoka = case rok
    when 476..1453:  'średniowiecze'
    when 1454..1918:  'nowożytność'
    when 1918..2018:  'współczesność'
end
```

Deklaracja słownika

```
slownik = {
    'one' => 1,
    'two' => 2,
    'three' => 3
}
```

Deklaracja słownika

```
slownik = {
    'one' => 1,
    'two' => 2,
    'three' => 3
}
```

Pusty słownik

```
slownik = { }

slownik = Hash.new(0)
```

Przetwarzanie słowników

```
slownik = { 1=> 'jeden', 2 => 'dwa' }
for  k in  slownik.keys
     puts '[' + k.to_s + ']=' + slownik[k].to_s
end
```

Plan wykładu

Składnia

```
{
    instrukcje
}
```

```
do
    instrukcje
end
```

Bloki z parametrem (parametrami)

{ | z1, z2 | *instrukcje* }

Zastosowania bloków

Przetwarzanie tablic

```
arr = ['czerwony', 'biały', 'zielony']
arr.each { | item | printf("Kolor %s\n", item) }
```

Zastosowania bloków

Przetwarzanie tablic

```
arr = ['czerwony', 'biały', 'zielony']
arr.each { | item | printf("Kolor %s\n", item) }
```

Przetwarzanie słowników

```
slownik.each { |k, v| puts "#{k} => #{v}" }
```

Zastosowania bloków

Przetwarzanie tablic

```
arr = ['czerwony', 'biały', 'zielony']
arr.each { | item | printf("Kolor %s\n", item) }
```

Przetwarzanie słowników

```
slownik.each { |k, v| puts "#{k} => #{v}" }
```

Inne przykłady

```
15.downto(0) { |i| printf("%04b\n", i) }
```

```
'xxx'.upto('xyz') { | i | puts i }
```

Ostatnie przykłady

```
[1,2,3,4].collect! { |i| 2**i }
```

```
[2, 4, 8, 16]
```

Ostatnie przykłady

```
[1,2,3,4].collect! { |i| 2**i }
```

```
[2, 4, 8, 16]
```

```
[1,2,3,4].delete_if { | i | i % 2 == 0 }
```

```
[1, 3]
```