Programowanie w Ruby

Wykład 3

Marcin Młotkowski

21 października 2016

Plan wykładu

Z poprzedniego wykładu

String to ciąg bajtów zwykle reprezentujących znaki.

Z poprzedniego wykładu

String to ciąg bajtów zwykle reprezentujących znaki.

Na jednym z forów

String to kolekcja zakodowanych danych, tj. surowe dane + kodowanie.

Badanie kodowania napisu

Ruby (co najmnniej 1.9.*)

```
__ENCODING__.name
=> "UTF-8"
"Ala ma kota".encoding
=> "UTF-8"
```

Plik źródłowy

```
# encoding: utf-8

alias :λ :lambda
π = Math::PI

hello_world = λ{|subject| "Hello, #{subject}! π is #{π}"}
```

Plik źródłowy

```
# encoding: utf-8

alias :λ :lambda
π = Math::PI

hello_world = λ{|subject| "Hello, #{subject}! π is #{π}"}
```

Kodowanie z Emacs'a

```
# -*- encoding: utf-8 -*-
```

Plan wykładu

Rodzaje zmiennych

- Lokalne
- Zmienne instancyjne (obiektu)
- Zmienne globalne
- Stałe

Zmienne lokalne

Nazwa zmiennej lokalnej

Nazwa zmiennej lokalnej zaczyna się od małej litery lub znaku podkreślenia _.

Zmienne lokalne

Nazwa zmiennej lokalnej

Nazwa zmiennej lokalnej zaczyna się od małej litery lub znaku podkreślenia _.

Zasięg zmiennej

Zmienna jest dostępna tylko w kontekście (w funkcji, w pętli, etc.) w którym została utworzona

Zmienne lokalne

Nazwa zmiennej lokalnej

Nazwa zmiennej lokalnej zaczyna się od małej litery lub znaku podkreślenia _.

Zasięg zmiennej

Zmienna jest dostępna tylko w kontekście (w funkcji, w pętli, etc.) w którym została utworzona

Zmienna poza zasięgiem

```
zmienna = 'zmienna'
def f
    puts zmienna
end
f
```

Zmienne globalne

Zmienne globalne zaczynają się od znaku $.

Zmienne globalne

Zmienne globalne zaczynają się od znaku $.

Przykład zmiennej globalnej

```
$zmienna = 'zmienna'
def f
    puts $zmienna
end
f
```

Stałe

Nazwa stałej

Nazwa stałej zaczyna się wielką literą.

Stałe

Nazwa stałej

Nazwa stałej zaczyna się wielką literą.

```
Const = 'stała'
# można, ale będzie ostrzeżenie:
Const = 'zmienna?'
```

Zmienne klasy i zmienne instancyjne

@zmienna_instancyjna
@@zmienna_klasy

Plan wykładu

Przypomnienie

{ | z1, z2 | *instrukcje* }

do | z1, z2 | *instrukcje* end

Wykonanie bloku

yield

Przykład

```
def run
    yield
end
```

Przykład

```
def run
    yield
end
```

```
run { puts 2+2 }
```

Bardziej skomplikowany przykład

```
def run
    yield 2, 2
    yield 3, 4
end
```

Bardziej skomplikowany przykład

```
def run
    yield 2, 2
    yield 3, 4
end
```

```
run { | x, y | puts x + y }
```

Jeszcze jeden przykład

```
def run
    yield 2, 2
end
```

```
z=2
run { |x, y| puts x + y + z }
```

Bloki to obiekty!

```
blok = proc { | x, y | puts x + y }
blok.call 2, 3
```

Lepiej: lambda

```
blok = lambda { puts 2+2 }
blok.call
```

Można też tak:

```
def fun (&block)
    block.call
end

fun { puts 'Cześć!!!' }
```

Uwagi

- Bloki to obiekty klasy Proc
- Można tworzyć bloki w stylu obiektowym

$$bl = Proc.new\ \{\ puts\ \text{'A kuku'}\ \}$$

ale to czasem jest zły pomysł! Również & czasem powoduje kłopoty

Różnice między proc a lambda

wrażliwość na liczbę argumentów i parametrów

Obiekty `proc` zwracają wyjątek jeśli liczba argumentów wywołania nie zgadza się z liczbą parametrów, blokom `lambda` to wszystko jedno. Brakujące argumenty to `nil`.

Bloki a instrukcja return

Źle

```
def boo
    Proc.new { return }
end
x = boo
x.call
```

Bloki a instrukcja return

Źle

```
def boo
    Proc.new { return }
end
x = boo
x.call
```

Dobrze

```
def boo
    lambda { return }
end
x = boo
x.call
```

Plan wykładu

Przykład

moduly.rb

```
module Matematyka
    def Matematyka.dodawanie(x, y)
        x+y
    end
    Pi = 3.1415
end
```

Przykład

moduly.rb

```
module Matematyka
    def Matematyka.dodawanie(x, y)
        x+y
    end
    Pi = 3.1415
end
```

plik.rb

```
require 'moduly'
puts Matematyka.dodaj(2, 2)
puts Matematyka::Pi
```

Dołączanie kodu

require

ładuje plik tylko raz, za pierwszym razem gdy jest wywoływana

load

ładuje pliki za każdym razem, gdy wykonanie programu dojdzie do tej instrukcji

Uwagi o modułach

- Nazwa modułu musi być pisana wielką literą
- W jednym pliku może być wiele modułów
- Moduły można zagnieżdżać
- W module można umieszczać instrukcje, które są wykonywane podczas włączania modułu

Bloki kodu — składnia

```
BEGIN {
    ciąg instrukcji
}
```

```
END {
    ciąg instrukcji
}
```

Semantyka bloków kodu

- Bloki BEGIN i END można umieszczać w każdym pliku z kodem źródłowym
- Bloki BEGIN są wykonywane podczas ładowania pliku; natomiast END przy zakończeniu programu

Plan wykładu

Przykłady komentarzy

Komentarz jednowierszowy

```
# to jest zwykły komentarz
```

Komentarz wielowierszowy

```
=begin
A to jest bardzo obszerny, wielowierszowy
komentarz
=end
```

Dokumentowanie modułów — RDoc

- Analizowany jest komentarz # na początku pliku oraz przed deklaracjami funkcji i modułów
- Generowanie dokumentacji:
  $ rdoc plik.rb
  Wynik jest w katalogu doc/

Formatowanie w RDoc

- #-- powoduje pomijanie komentarza aż do #++
- #= generuje nagłówek, np.
  #= USAGE
- #== również generuje nagłówek
- Można używać html-a
- Wiele innych znaczników

Jeszcze inny komentarz

```
=begin rdoc

=end
```

Doc dla RDoc

http://rdoc.sourceforge.net/doc/files/README.html