Programowanie obiektowe

Język programowania Ruby

Marcin Młotkowski

12 kwietnia 2018

Plan wykładu

Dlaczego Ruby

Ortodoksyjny język obiektowy ("wszystko jest obiektem").

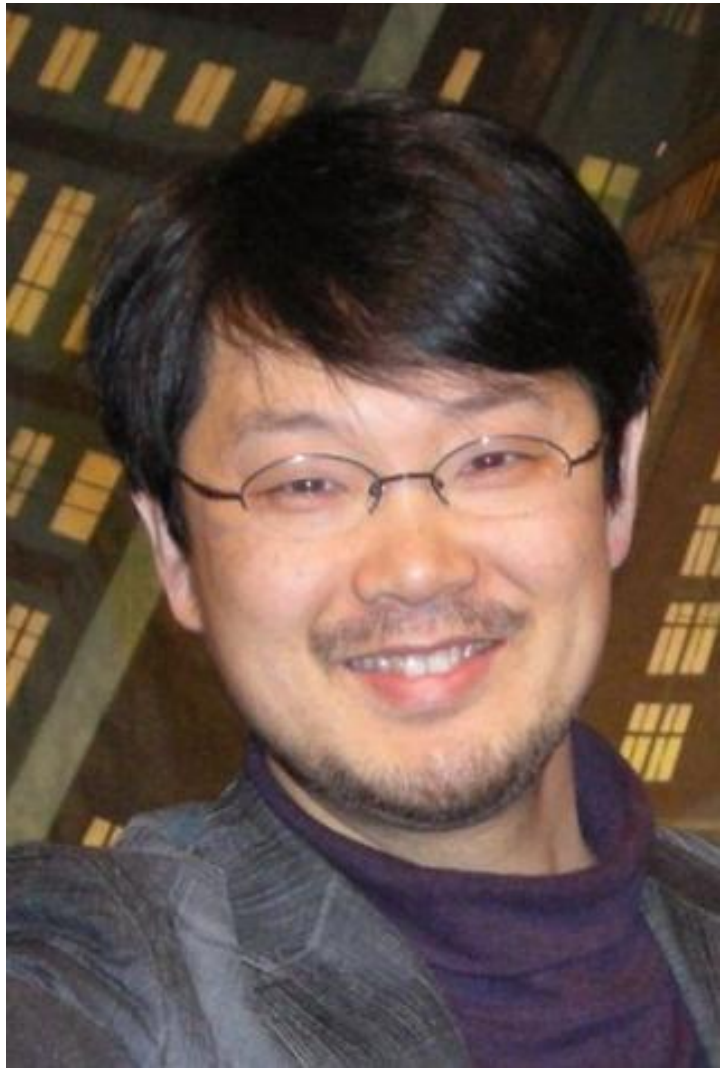
Ruby

Główne cechy języka

- Język skrajnie obiektowy;
- dynamiczny system typów (skryptowy);
- mechanizm domieszkowania klas (mix–ins);
- cechy języków funkcjonalnych;
- kontynuacje;
- i sporo innych ...

Autorstwo

Yukihiro Matsumoto (Matz)



Uruchamianie programów

Tryb interaktywny

```
$ ruby
puts "A kuku!"
^D
$ irb # albo jirb
irb(main):001:0>
```

Uruchamianie programów

Tryb interaktywny

```
$ ruby
puts "A kuku!"
^D
$ irb # albo jirb
irb(main):001:0>
```

Wykonywanie programów

```
$ ruby plik.rb
```

Dokumentacja

```
$ ri Fixnum
```

Plan wykładu

Klasa Fixnum, Bignum, Float

- liczby są prawdziwymi obiektami;
- dostępne są standardowe operatory arytmetyczne

Literały

- literały proste
  'Ala ma kota'
- literały bardziej skomplikowane
  "Ala ma #{2+2} koty"
- i jeszcze inne

Operowanie napisami

- +
- mnóstwo innych

Przedziały

Literały:

1..10 $\rightarrow$ 1, 2, ..., 10
'a'...'d' $\rightarrow$ 'a', 'b', 'c'

Zastosowanie przedziałów

Test zawierania:

(1..10) === 5 $\rightarrow$ true

('a'..'j') === 'z' $\rightarrow$ false

Deklarowanie i odwołania do tablic

zwierzątka = ['kotek', 1024, 'piesek', 3.1415]

zwierzątka[2] $\rightarrow$ 'piesek'

Deklarowanie tablic asocjacyjnych

słownik = {
'jeden' => 'one',
'dwa' => 'two',
'trzy' => 'three'
}
słownik['dwa'] → 'two'

Konwencje w Ruby

$zmienna	zmienna globalna
@zmienna	zmienna obiektu
@@zmienna	zmienna klasy (statyczna)
Zmienna	stała, nazwa klasy lub modułu
zmienna	zmienna lokalna, parametry i nazwy metod

Plan wykładu

Definicja prawdy

Prawdziwa jest każda wartość, która nie jest nil ani false.

Spójniki logiczne

and, &&, or, ||, not, !

Wartość wyrażenia logicznego

Wartością wyrażenia logicznego jest wartość tego argumentu, który determinuje prawdę lub fałsz.

Wartość wyrażenia logicznego

Wartością wyrażenia logicznego jest wartość tego argumentu, który determinuje prawdę lub fałsz.

nil and true	nil
false and true	false
1024 and false	false
1024 and nil	nil
1024 and "napis"	"napis"

Zastosowanie

lista[klucz] ||= [ ]

jest równoważne

lista[klucz] = lista[klucz] || [ ]

Operatory logiczne

==	sprawdzenie równości
<, <=, >=, >	standardowe operatory arytmetyczne

Operator defined?

defined? 1	"expression"
defined? niezdefinowana	nil
defined? printf	"method"

Plan wykładu

Instrukcje, a właściwie wyrażenia

Większość instrukcji można traktować jak wyrażenia.

Instrukcja (wyrażenie) przypisania

```
x = 2 + 2
```

Instrukcja (wyrażenie) przypisania

```
x = 2 + 2
```

```
a = b = c = 2 * 2
```

Instrukcja (wyrażenie) przypisania

```
x = 2 + 2

a = b = c = 2 * 2

a, b = b, a
```

Schemat instrukcji złożonej

Instrukcja

    ...

end

Instrukcja if , 1. postać

if *warunek*
    instrukcja
end

Instrukcja if , 2. postać

```
if warunek
    instrukcja
else
    instrukcja
end
```

Instrukcja if , 2. postać

if *warunek*
    instrukcja
else
    instrukcja
end

if *warunek* then *instrukcja*
else *instrukcja*
end

Instrukcja if , 2. postać

```
if warunek
    instrukcja
else
    instrukcja
end

if warunek then instrukcja
else instrukcja
end

if warunek: instrukcja
else instrukcja
end
```

Instrukcja if , 3. postać

```
if warunek
    instrukcja
elsif warunek
    instrukcja
else
    instrukcja
end
```

Instrukcja if w roli wyrażenia

```
abs = if x < 0
          -x
      else
          x
      end
```

Skrócona postać instrukcji if

```
print suma if suma > 0

print suma unless suma == 0
```

Instrukcja case, 1. postać

```
przestepny = case
             when rok % 400 == 0: true
             when rok % 100 == 0: false
             else rok % 4 == 0
             end
```

Instrukcja case , 2. postać

```
case polecenie
when "exit", "quit"
    exit
when "run"
    init()
    start()
else
    print "Nieznane polecenie #{polecenie}"
```

Pętle

```
while warunek
    ...
end
```

```
unless warunek
    ...
end
```

Pętle

while *warunek*

...

end

unless *warunek*

...

end

Postać skrócona

a *= 2 while $a < 100$

delta = delta - 10 unless $delta < 0$

Powtarzanie

```
3.times do
    print "Hop!\n"
end
```

```
3.times do |x|
    print "Hop!\n"
end
```

```
0.upto(9) do |x|
    print " ", x
end
```

Iteracja po kolekcjach

```
[ 2, 3, 5, 7, 11].each { | val | print val, " "}
```

Plan wykładu

Wszystkie klasy dziedziczą po klasie `Object`.

Klasa reprezentująca książkę

```
class Ksiazka
    def initialize(tytul, autor, wydawca)
        @tytul = tytul
        @autor = autor
        @wydawca = wydawca
    end
end
```

Tworzenie obiektu

obiekt = Ksiazka.new('Pan Tadeusz', 'Mickiewicz', 'Toruń')
print obiekt.to_s $\rightarrow$ '#<Ksiazka:0x3f4b21>'

Implementacja metod

```
class Ksiazka
    def initialize(tytul, autor, wydawca)
        @tytul = tytul
        @autor = autor
        @wydawca = wydawca
    end

    def tytul
        @tytul
    end
end
```

Implementacja metod

```
class Ksiazka
    def initialize(tytul, autor, wydawca)
        @tytul = tytul
        @autor = autor
        @wydawca = wydawca
    end

    def tytul
        @tytul
    end
end
```

Odwołanie do metody

```
k = Ksiazka.new('Pan Tadeusz', 'Mickiewicz', 'Toruń')
k.tytul()
k.tytul
```

Metody "specjane"

```
class Temperatura
    def initialize(temp)
        @celsjusz = temp
    end



end
```

Metody "specjane"

```
class Temperatura
    def initialize(temp)
        @celsjusz = temp
    end
    def fahrenheit
        32.0 + (9.0/5.0)*@celsjusz
    end



end
```

Metody "specjane"

```
class Temperatura
    def initialize(temp)
        @celsjusz = temp
    end
    def fahrenheit
        32.0 + (9.0/5.0)*@celsjusz
    end
    def fahrenheit=(temp)
        @celsjusz = (5.0/9.0)*(temp - 32)
    end
end
```

Wirtualny atrybut

temp = Temperatura.new(36.6)

temp.fahrenheit $\rightarrow$ 97.88

temp.fahrenheit = 32

Składnia

```
class Cwiczenia < Ksiazka
    def initialize(tytul, autor, wydanie, cena)
        super(tytul, autor, wydanie)
        @cena = cena
    end
    def to_s
        super+ @cena.to_s
    end
end
```

Pola

Pola obiektów są prywatne.

Rodzaje metod

metody publiczne domyślnie metody są publiczne, z wyjątkiem `initialize`;

metody chronione są to metody, do których dostęp mają tylko obiekty tej samej klasy i klas potomnych;

metody prywatne dostęp jest jedynie w kontekście bieżącego obiektu.

Definiowanie dostępu do metod

```
class Klasa
    def metoda1
    end
private
    def metoda2
    end
    def metoda3
    end
protected
    def metoda4
    end
    def metoda5
    end
end
```

Zmiennna klasy

```
class Klasa
    @@licznik = 0
    def initialize
        @@licznik += 1
    end
end
```

Metody statyczne

```
class Klasa
    @@licznik = 0
    def initialize
        @@licznik += 1
    end
    def Klasa.info
        print "Jest #{@@licznik} obiektów tej klasy"
    end
end
```

Jak to zrobić

Już zadeklarowane klasy można rozszerzać

Jak to zrobić

Już zadeklarowane klasy można rozszerzać

Przykład

```
class Fixnum
    def next
        self+ 1
    end
end
```