

Programowanie obiektowe

Wykład 8.

Marcin Młotkowski

16 kwietnia 2024

Plan wykładu

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa String)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 Wyrażenia i instrukcje
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

Plan wykładu

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa `String`)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 Wyrażenia i instrukcje
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

Dlaczego Ruby

Ortodoksyjny język obiektowy ("wszystko jest obiektem").

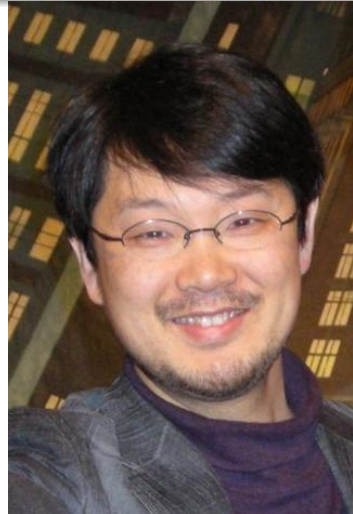
Ruby

Główne cechy języka

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

Autorstwo

Yukihiro Matsumoto (Matz)



Uruchamianie programów

Tryb interaktywny

\$ irb # albo jirb

irb(main):001:0>

Uruchamianie programów

Tryb interaktywny

```
$ irb # albo jirb
```

```
irb(main):001:0>
```

Wykonywanie programów

```
$ ruby plik.rb
```


Dokumentacja

\$ ri Integer

\$ ri String

Plan wykładu

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa String)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 Wyrażenia i instrukcje
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

Klasa Integer (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:

$1..10 \rightarrow 1, 2, \dots, 10$

$'a'...'d' \rightarrow 'a', 'b', 'c'$

Zastosowanie przedziałów

Test zawierania:

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

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

Deklarowanie i odwołania do tablic

```
zwierzatka = [ 'kotek', 1024, 'piesek', 3.1415 ]  
zwierzatka[2] → '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

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa `String`)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 Wyrażenia i instrukcje
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

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.

<code>nil and true</code>	<code>nil</code>
<code>false and true</code>	<code>false</code>
<code>1024 and false</code>	<code>false</code>
<code>1024 and nil</code>	<code>nil</code>
<code>1024 and "napis"</code>	<code>"napis"</code>

Zastosowanie

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

jest równoważne

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


Operatory logiczne

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

Operator **defined?**

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

Plan wykładu

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa `String`)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 **Wyrażenia i instrukcje**
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

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 a < 100  
    a *= 2  
end
```

```
unless delta < 0  
    delta -= 10  
end
```

Pętle

```
while a < 100  
    a *= 2  
end
```

Postać skrócona

```
a *= 2 while a < 100
```

```
unless delta < 0  
    delta -= 10  
end
```

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

Inne operacje

```
[ 2, 3, 5, 7, 11].select { | val | val > 4 }  
[ 2, 3, 5, 7, 11].map { | val | 2**val }
```

Plan wykładu

- 1 Wstęp
- 2 Typy wbudowane
 - Typy numeryczne
 - Łańcuchy znaków (klasa `String`)
 - Przedziały
 - Tablice i tablice asocjacyjne
 - Nazwy zmiennych
- 3 Wyrażenia logiczne
- 4 Wyrażenia i instrukcje
 - Przypisania
 - Instrukcje warunkowa
 - Pętle
- 5 Klasy i obiekty
 - Deklaracja klasy

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 → '#<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 "specjalne"

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

Metody "specjalne"

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

Metody "specjalne"

```
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 → 97.88  
temp.fahrenheit = 32
```

Dalsze rozszerzenie

```
def kelvin  
    @celsjusz + 273.15  
end
```

Dalsze rozszerzenie

```
def kelvin
  @celsjusz + 273.15
end

def reaumur
  @celsjusz * 0.8
end
```

Składnia

```
class Cwiczenia < Ksiazka
  def initialize(tytuł, autor, wydanie, cena)
    super(tytuł, 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
```