EPI: Interfejs Graficzny 2009/2010 Podstawy Rubiego

Aleksander Pohl

1 czerwca 2010

Plan prezentacji

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie
- 1999 – bardziej popularny w Japonii niż Python

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie
- 1999 – bardziej popularny w Japonii niż Python
- 2000 – pierwsza lista dyskusyjna

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie
- 1999 – bardziej popularny w Japonii niż Python
- 2000 – pierwsza lista dyskusyjna
- 2004 Dania – Ruby on Rails, David Heinemeier Hansson

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie
- 1999 – bardziej popularny w Japonii niż Python
- 2000 – pierwsza lista dyskusyjna
- 2004 Dania – Ruby on Rails, David Heinemeier Hansson
- 2005, 2006 O'Reilly – sprzedano więcej książek na temat Rubiego niż na temat Perla czy Pythona

Historia Rubiego

- 1993 Japonia – Yukihiro Matsumoto San
- 1995 – pierwsze wydanie
- 1999 – bardziej popularny w Japonii niż Python
- 2000 – pierwsza lista dyskusyjna
- 2004 Dania – Ruby on Rails, David Heinemeier Hansson
- 2005, 2006 O'Reilly – sprzedano więcej książek na temat Rubiego niż na temat Perla czy Pythona
- 2010 – planowane 3 wydanie frameworku Ruby on Rails (połączonego z Merbem)

Cechy języka

- interpretowany

Cechy języka

- interpretowany
- dynamicznie typizowany

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania
- *garbage collector*

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania
- *garbage collector*
- dziedziczenie jednobazowe

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania
- *garbage collector*
- dziedziczenie jednobazowe
- mechanizm wyjątków

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania
- *garbage collector*
- dziedziczenie jednobazowe
- mechanizm wyjątków
- bloki i domknięcia

Cechy języka

- interpretowany
- dynamicznie typizowany
- silnie typizowany
- w 100% zorientowany obiektowo
- wspiera funkcjonalny styl programowania
- *garbage collector*
- dziedziczenie jednobazowe
- mechanizm wyjątków
- bloki i domknięcia
- metaprogramowanie

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)
- brak wymogów co do wcięć (oczywiście właściwe wcięcia poprawiają czytelność kodu)

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)
- brak wymogów co do wcięć (oczywiście właściwe wcięcia poprawiają czytelność kodu)
- brak deklaracji typów – wystarcza inicjowanie zmiennych

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)
- brak wymogów co do wcięć (oczywiście właściwe wcięcia poprawiają czytelność kodu)
- brak deklaracji typów – wystarcza inicjowanie zmiennych
- 1-linijkowe komentarze zaczynają się znakiem #

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)
- brak wymogów co do wcięć (oczywiście właściwe wcięcia poprawiają czytelność kodu)
- brak deklaracji typów – wystarcza inicjowanie zmiennych
- 1-linijkowe komentarze zaczynają się znakiem #
- specjalny *obiekt* reprezentujący wartość pustą zwany `nil`

Czego można się spodziewać?

- brak średników (pod warunkiem, że nie umieszczasz wielu poleceń w jednej linii, co jednak jest odradzane)
- brak wymogów co do wcięć (oczywiście właściwe wcięcia poprawiają czytelność kodu)
- brak deklaracji typów – wystarcza inicjowanie zmiennych
- 1-linijkowe komentarze zaczynają się znakiem #
- specjalny *obiekt* reprezentujący wartość pustą zwany `nil`
- każde wyrażenie ewaluuje się do wartości (również `if`, `case`, itp.)

Plan prezentacji

"Hello World" w Rubim

- Rozpocznij interaktywną sesję Rubiego wpisując w linii poleceń:

```
$ irb
>>
```

"Hello World" w Rubim

- Rozpocznij interaktywną sesję Rubiego wpisując w linii poleceń:

```
$ irb
```

```
>>
```

- "Witaj Świecie" w linii poleceń Rubiego:

```
>> puts "Witaj Świecie!"
>> print "Witaj Świecie!"
```

"Hello World" w Rubim

- Rozpocznij interaktywną sesję Rubiego wpisując w linii poleceń:

```
$ irb
```

```
>>
```

- "Witaj Świecie" w linii poleceń Rubiego:

```
>> puts "Witaj Świecie!"
>> print "Witaj Świecie!"
```

- Jaka jest różnica pomiedzy `puts` oraz `print`?

"Hello World" w Rubim

- Rozpocznij interaktywną sesję Rubiego wpisując w linii poleceń:

```
$ irb
>>
```

- "Witaj Świecie" w linii poleceń Rubiego:

```
>> puts "Witaj Świecie!"
>> print "Witaj Świecie!"
```

- Jaka jest różnica pomiedzy `puts` oraz `print`?
- A teraz "Witaj Świecie" w wersji *enterprise*:

```
>> name="Mistrzu"
>> puts "Witaj "+name+"!!!"
```

"Hello World" w Rubim

- Rozpocznij interaktywną sesję Rubiego wpisując w linii poleceń:

```
$ irb
>>
```

- "Witaj Świecie" w linii poleceń Rubiego:

```
>> puts "Witaj Świecie!"
>> print "Witaj Świecie!"
```

- Jaka jest różnica pomiedzy `puts` oraz `print`?
- A teraz "Witaj Świecie" w wersji *enterprise*:

```
>> name="Mistrzu"
>> puts "Witaj "+name+"!!!"
```

- Aby opuścić sesję `irb` wrowadź `quit`:

```
>> quit
```

"Hello World" jako skrypt

- Otwórz swój ulubiony edytor testu i utwórz skrypt Rubiego o nazwie `hello.rb` :

```
puts "Witaj Świecie!"
```

"Hello World" jako skrypt

- Otwórz swój ulubiony edytor testu i utwórz skrypt Rubiego o nazwie `hello.rb` :

```
puts "Witaj Świecie!"
```

- Zapisz skrypt i wywołaj go wpisując:

```
$ ruby hello.rb
```

"Hello World" jako skrypt

- Otwórz swój ulubiony edytor testu i utwórz skrypt Rubiego o nazwie `hello.rb` :

```
puts "Witaj Świecie!"
```

- Zapisz skrypt i wywołaj go wpisując:

```
$ ruby hello.rb
```

- Popraw skrypt tak, aby zapytał cię o imię:

```
puts "Jak masz na imię?"
name = gets
puts "Witaj "+name+"!!!"
```

Definiowanie funkcji

- Zdefiniujmy prostą funkcję, która jako argument będzie przyjmowała imię i będzie zwracała łańcuch ”Hello ”+name:

```
def say_hello(name)
  "Witaj "+name
end
```

Definiowanie funkcji

- Zdefiniujmy prostą funkcję, która jako argument będzie przyjmowała imię i będzie zwracała łańcuch ”Hello ”+name:

```
def say_hello(name)
  "Witaj "+name
end
```

- Funkcja jako swój rezultat zwraca wartość ostatniego ewaluowanego wyrażenia. Możesz jednak bezpośrednio użyć słowa kluczowego return:

```
def say_hello(name)
  return "Witaj "+name
end
```

Wywoływanie funkcji

- W Rubim możesz wywołać funkcję w zwykły sposób umieszczając argumenty w nawiasach:

```
say_hello("Janek")
```

Wywoływanie funkcji

- W Rubim możesz wywołać funkcję w zwykły sposób umieszczając argumenty w nawiasach:
  ```
  say_hello("Janek")
  ```
- Jednakże nawiasy mogą być opuszczone, o ile nie prowadzi to do niejednoznaczności:
  ```
  say_hello "Janek"
  ```

Wywoływanie funkcji

- W Rubim możesz wywołać funkcję w zwykły sposób umieszczając argumenty w nawiasach:
  ```
  say_hello("Janek")
  ```
- Jednakże nawiasy mogą być opuszczone, o ile nie prowadzi to do niejednoznaczności:
  ```
  say_hello "Janek"
  ```
- Teraz zmodyfikuj skrypt "hello.rb", tak aby korzystał z funkcji `say_hello`!

Argumenty funkcji

Argumenty funkcji mogą być:

- opcjonalne

```
def say_hello(name="Świecie")
  "Witaj " + name
end

say_hello "Andrzej"
# "Witaj Andrzej"
say_hello
# "Witaj Świecie"
```

Argumenty funkcji

Argumenty funkcji mogą być:

- opcjonalne

```ruby
def say_hello(name="Świecie")
  "Witaj " + name
end

say_hello "Andrzej"
# "Witaj Andrzej"
say_hello
# "Witaj Świecie"
```

- w postaci par „klucz – wartość”

```ruby
def say_hello(args)
  "Witaj " + args[:name] + " " + args[:surname]
end

say_hello :name => "Jan", :surname => "Kowalski"
# "Witaj Jan Kowalski"
```

Uwaga na temat nazewnictwa

- zmienne oraz funkcje są zawsze zapisywane `z_użyciem_znaku_podkreślenia`

Uwaga na temat nazewnictwa

- zmienne oraz funkcje są zawsze zapisywane `z_użyciem_znaku_podkreślenia`
- stałe zaczynają się od dużej litery, najlepiej `W_CAŁOŚCI_KAPITALIKAMI`

Uwaga na temat nazewnictwa

- zmienne oraz funkcje są zawsze zapisywane `z_użyciem_znaku_podkreślenia`
- stałe zaczynają się od dużej litery, najlepiej `W_CAŁOŚCI_KAPITALIKAMI`
- klasy i moduły zapisywane są z użyciem `NotacjiWielbłądziej`

Uwaga na temat nazewnictwa

- zmienne oraz funkcje są zawsze zapisywane `z_użyciem_znaku_podkreślenia`
- stałe zaczynają się od dużej litery, najlepiej `W_CAŁOŚCI_KAPITALIKAMI`
- klasy i moduły zapisywane są z użyciem `NotacjiWielbłądziej`
- wypróbujmy to w `irb`:

```
>> HELLO = "hello"
>> HELLO = "goodbye"
```

Plan prezentacji

Sinatra – microframework

- Zamiast pisać w konsoli, możemy użyć prostego frameworku - pozwoli nam oglądać wyniki w przeglądarce

Sinatra – microframework

- Zamiast pisać w konsoli, możemy użyć prostego frameworku - pozwoli nam oglądać wyniki w przeglądarce
- `gem install sinatra sinatra-reloader`

Sinatra – microframework

- Zamiast pisać w konsoli, możemy użyć prostego frameworku - pozwoli nam oglądać wyniki w przeglądarce
- `gem install sinatra sinatra-reloader`

- `vim app.rb`

```ruby
# app.rb
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  "Witaj Świecie"
end
```

Sinatra – microframework

- Zamiast pisać w konsoli, możemy użyć prostego frameworku - pozwoli nam oglądać wyniki w przeglądarce
- `gem install sinatra sinatra-reloader`

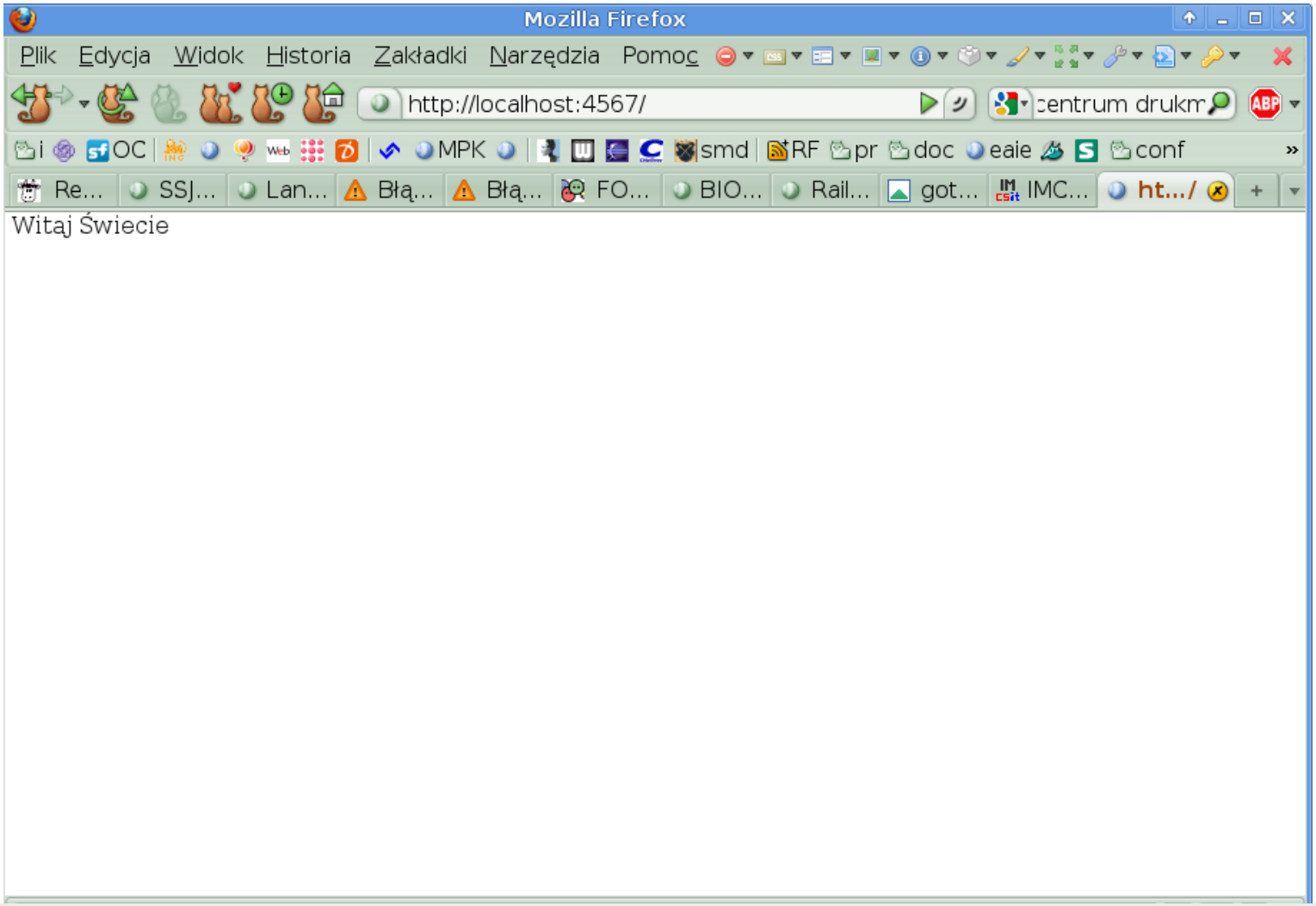
- `vim app.rb`

```ruby
# app.rb
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  "Witaj Świecie"
end
```

- `ruby app.rb`

Sinatra – screenshot



Sinatra – formularz (1)

- app1.rb

```
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  erb :index
end
```

Sinatra – formularz (1)

- app1.rb

```
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  erb :index
end
```

- mkdir views

Sinatra – formularz (1)

- app1.rb

```
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  erb :index
end
```

- mkdir views
- views/index.erb

```
Wprowadź swoje imię:
<form method="post">
  <input type="text" name="name"/>
</form>
```

Sinatra – formularz (2)

- app1.rb - cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

Sinatra – formularz (2)

- app1.rb - cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

- views/result.erb

```
<%= @message %>
```

Sinatra – formularz (2)

- app1.rb - cd.

```
post '/' do
  @message = "Witaj " + params[:name]
  erb :result
end
```

- views/result.erb

```
<%= @message %>
```

- views/layout.erb

```
<html>
  <body style="width: 900px;margin: auto">
    <h2>Aplikacja formularz</h2>
    <div>
      <%= yield %>
    </div>
  </body>
</html>
```

Pytania

PYTANIA?