

# EPI: Interfejs Graficzny 2011/2012

## Podstawy Rubiego

Aleksander Pohl

18 października 2011

# Plan prezentacji

Hello World

Sinatra

Zadania

# Plan prezentacji

Hello World

Sinatra

Zadania

## ”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 pomiędzy `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 pomiędzy 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 pomiędzy puts oraz print?

- ▶ A teraz "Witaj Świecie" w wersji *enterprise*:

```
>> name="Mistrzu"
```

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

- ▶ Aby opuścić sesję irb wprowadź quit:

```
>> quit
```

# "Hello World" jako skrypt

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

```
# encoding: utf-8  
puts "Witaj Świecie!"
```

# "Hello World" jako skrypt

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

```
# encoding: utf-8  
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` :

```
# encoding: utf-8  
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

- ▶ Zdefiniujemy prostą funkcję, która jako argument będzie przyjmowała imię i będzie zwracała łańcuch "Witaj "+name:

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

# Definiowanie funkcji

- ▶ Zdefiniujmy prostą funkcję, która jako argument będzie przyjmowała imię i będzie zwracała łańcuch "Witaj "+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

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

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

- ▶ w postaci par „klucz – wartość”

```
def say_hello(person)  
  "Witaj " + person[:name] + " " + person[: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 Notacji Wielbłędziej

## 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 Notacji Wielbłędziej
- ▶ wypróbujmy to w irb:  

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

# Plan prezentacji

Hello World

Sinatra

Zadania

# 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`

- ▶ `app.rb`

```
# encoding: utf-8
# 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`

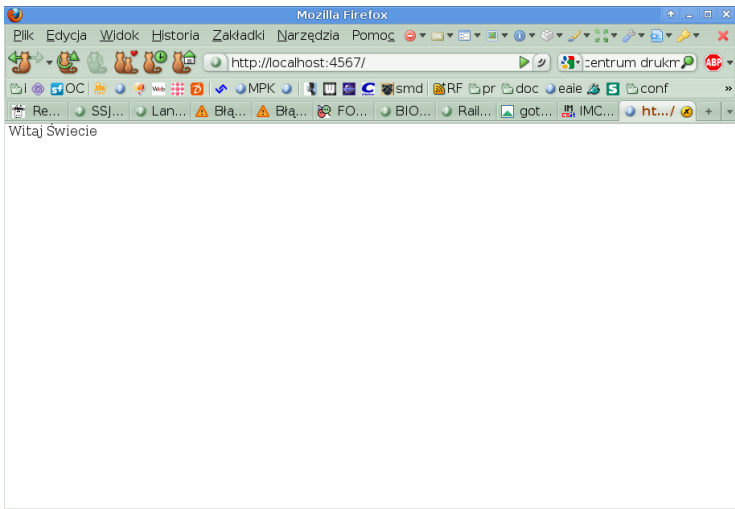
- ▶ `app.rb`

```
# encoding: utf-8
# app.rb
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader' if development?

get '/' do
  "Witaj Świecie"
end
```

- ▶ `ruby app.rb -p 3300`

# 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>
```

# Plan prezentacji

Hello World

Sinatra

Zadania

# Sprawdzanie kodu pocztowego

Napisz aplikację, która w formularzu akceptuje jeden argument będący napisem. Aplikacja powinna generować komunikat „Poprawny adres”, jeśli napis składa się z poprawnego kodu pocztowego, po którym następuje nazwa miasta pisana z wielkiej litery. W przeciwnym razie powinien być generowany komunikat „Niepoprawny adres”.

# Konwersja reprezentacji liczb

Napisz aplikację, która zamienia liczbę zapisaną w jednym systemie pozycyjną, na liczbę w innym systemie pozycyjnym.

Aplikacja powinna definiować formularz z trzema parametrami

- ▶ łańcuch znaków reprezentujący liczbę podlegającą konwersji
- ▶ podstawa systemu, w którym zapisana jest przekazana liczba
- ▶ podstawa systemu, w którym ma zostać zwrócony wynik

# Kalkulator

Napisz aplikację typu kalkulator – powinna ona definiować dwa pola tekstowe, w których można wprowadzić wartości liczbowe oraz pole selekcji, w którym możemy określić operację, która ma zostać wykonana. Jako wynik przesłania formularza powinien być generowany wynik zastosowania operacji do przekazanych argumentów. Zwróć uwagę na przypadki szczególne (np. dzielenie przez zero).

# Zliczanie słów

Napisz aplikację, która posiada formularz z polem tekstowym. Wynikiem działania aplikacji powinno być podsumowanie częstości występowania poszczególnych słów w przekazanym tekście. Podsumowanie to powinno być posortowane w kolejności malejącej liczby wystąpień i zwracać 20 najczęstszych wyników. Podsumowanie nie powinno brać pod uwagę wielkości liter w tekście.

## Więcej zadań

- ▶ try ruby  
`http://tryruby.org`
- ▶ przykładowe zadania  
`apohllo.pl/dydaktyka/interfejsy-graficzne/zadania-ruby`

# Pytania

PYTANIA?