Programowanie w Ruby

Wykład 5

Marcin Młotkowski

5 listopada 2018

Plan wykładu

Wyrażenia regularne w życiu codziennym

Przykłady

*.cpp
plik?.rb
listazadan.pdf
...

Przykładowe zadanie

Wyszukać w tekście adresy email.

Dodatkowe informacje

W adresie używane są wyłącznie małe litery, kropki i znak @, np. guest@foo.com

Rozwiązanie

```
[a-z]+@([a-z]+\.)+[a-z]+
```

Objaśnienie:

- [a-z] oznacza dowolną literę od a do z
- [a-z]+ oznacza co najmniej jedną (lub więcej) literę od a do z
- @ oznacza @
- \. oznacza kropkę
- ([a-z]+\.)+ oznacza ciąg liter zakończony kropką, powtarzający się co najmniej raz

Wyrażenia regularne w Ruby

Utworzenie wyrażenia

```
reg = /[a-z]+@([a-z]+\.)+[a-z]+/
```

Wyrażenia regularne w Ruby

Utworzenie wyrażenia

```
reg = /[a-z]+@([a-z]+\.)+[a-z]+/
```

Dopasowanie wzorca

```
if 'SPAM_imie@mail.info_USUNTO' =~ reg
    puts $`, $&, $'
end
```

=~ dopasowuje wzorzec do tekstu, wynikiem jest pozycja dopasowania lub nil

Wyniki dopasowania

$-variables

Wyniki dopasowania są w tzw. $–variables:

- $& fragment napisu dopasowany do wzorca
- $` fragment napisu przed wzorcem
- $' fragment napisu po wzorcu

Wynik działania programu

Program

```
reg = /[a-z]+@([a-z]+\.)+[a-z]+/
if 'SPAM_imie@mail.info_USUNTO' =~ reg
    puts $`, $&, $'
end
```

Wynik działania

```
SPAM_
imie@mail.info
_USUNTO
```

Wyszukiwanie wielu wzorców

Zadanie

Parsowane napisów postaci

rrrrmmdd

na przykład

20101104

Implementacja wzorca

```
patt = %r{(\d\d\d\d)(\d\d)(\d\d)}
```

Objaśnienia

- `%r{ ... }` jest inną formą definiowania wyrażenia regularnego
- `\d` oznacza cyfrę 0-9
- `()` oznacza grupowanie

Implementacja wzorca

```
patt = %r{(\d\d\d\d)(\d\d)(\d\d)}
```

Objaśnienia

- `%r{ ... }` jest inną formą definiowania wyrażenia regularnego
- `\d` oznacza cyfrę 0-9
- `()` oznacza grupowanie

Użycie

```
if '20101104' =~ patt
    puts "#{$&} = (#{$1.to_i}, #{$2.to_i}, #{$3.to_i})"
end
```

Wynik

```
20101104 = (2010, 11, 4)
```

Jeszcze jeden przykład

Przechodzimy na euro

```
str = '234 zł dodać 123 zł daje razem 357 zł'
str.gsub!(/zł/, 'eu')
```

```
234 eu dodać 123 eu daje razem 357 eu
```

Jak zamienić kwotę

```
str = '234 zł dodać 123 zł daje razem 357 zł'
str = str.gsub(/\d+/) { ($&.to_i/3.62).to_s }
```

```
64.6408839779006 eu dodać 33.9779005524862 eu daje razem
98.6187845303867 eu
```

Inne wyrażenia regularne

- Powtórzenia: r*, r+, r?, r{m, n}
- | oznacza alternatywę
- Klasy znaków: \d, \s, \w, ...
- ^ i $
- [aeiou]

Wyrażenia regularne na obiektowo

```
re = Regexp.new('(\d\d):(\d\d)')
res = re.match('12:45') # MatchData
puts res[0], res[1], res[2]
```

A jak wyszukać wiele wystąpień

```
"Ala ma kota".scan(/a/)
```

```
["a", "a", "a"]
```

Plan wykładu

Dawno, dawno temu

Reakcja na błędy, Turbo Pascal

```
read(f, buf);
if IOResult <> 0 then
```

Wyjątki

Mechanizm przekazania kontroli do innego miejsca w programie wraz z wartością

Wyjątki

Mechanizm przekazania kontroli do innego miejsca w programie wraz z wartością

Wyjątki w Ruby'm

- wyjątek to obiekt klasy Exception lub pochodnej
- Wyjątki można zgłaszać (raise)
- Wyjątki można łapać i obsługiwać

Zgłaszanie wyjątków

```
def foo(arg)
    if !arg.instance_of? Fixnum
        raise 'Nieprawidłowy argument'
    end
    return arg
end
```

Zgłaszanie wyjątków

Składnia instrukcji raise

raise

zgłasza wyjątek $! lub RuntimeError, gdzie $! oznacza ostatnio wywołany wyjątek

Zgłaszanie wyjątków

Składnia instrukcji raise

raise

zgłasza wyjątek $! lub RuntimeError, gdzie $! oznacza ostatnio wywołany wyjątek

raise 'komunikat'

tworzy obiekt `RuntimeError` ze wskazanym komunikatem, równoważne raise RuntimeError.new("komunikat")

Zgłaszanie wyjątków

Składnia instrukcji raise

raise

zgłasza wyjątek $! lub RuntimeError, gdzie $! oznacza ostatnio wywołany wyjątek

raise 'komunikat'

tworzy obiekt `RuntimeError` ze wskazanym komunikatem, równoważne raise RuntimeError.new("komunikat")

raise Klasa, 'komunikat', traceback

traceback jest śladem wywołań w postaci tablicy typu `String`

Obsługa wyjątków

składowe funkcjonalne obsługi wyjątków

- strefa krytyczna
- obsługa wyjątku (wyjątków)
- instrukcje wykonywane gdy wyjątku nie będzie
- instrukcje, które będą wykonane zawsze

Składnia

```
begin
    asm = compile(source)
rescue SyntaxError
    $stderr.print 'Error: ' + $!
rescue UndefinedVariableError => var
    raise var
else
    asm.save
ensure
    stream.close
end
```

Inny przykład

Zapisanie błędów programu

```
begin
    main
rescue
    save_to_file($!)
end
```

Wyjątki — uzupełnienie

retry

ponownie wywołuje blok begin — end

catch (:nazwa) do instrukcje end

Definiuje i wykonuje ciąg instrukcji, być może zagnieżdżony

throw (:nazwa)

Zwija stos wywołań aż do momentu odnalezienia bloku o nazwie :nazwa

Plan wykładu

Podejście najprostsze

```
file = File.open(fname, 'w')
...
file.close
```

Pliki po Rubiemu

Składnia

```
open(fname, 'r') do | fh | # fh.class == File
    przetwarzanie pliku
end
```

Pliki po Rubiemu

Składnia

```
open(fname, 'r') do | fh | # fh.class == File
    przetwarzanie pliku
end
```

Metody klasy File

gets	zwraca kolejny wiersz lub nil
each_line	blok jednoargumentowy
each_byte	blok jednoargumentowy

Przykłady

Przetwarzanie pliku tekstowego

```
open("plik.txt", 'r') do | fh |
    fh.each_line { | line | puts line }
end
```

Przetwarzanie strumienia tekstowego

```
require 'open-uri'

open('http://www.ii.uni.wroc.pl') do | fh |
    fh.each_line { | line | puts line }
end
```

Inny przykład

```
require 'open-uri'

open('http://www.ii.uni.wroc.pl') do | fh |
    fh.each_line { | line | puts line }
end
```