Parrot i Perl 6 - Wprowadzenie

Prezentacja przygotowana przez

Bartka Jakubskiego <migo@pld-linux.org>

i pierwotnie zaprezentowana na

Jesieni Linuksowej 2004 (Ustroń 8-10.10.2004).

Zimowisko TLUG (Puck 15.1.2005)

Bartek Filipowicz <bfilipow1@wp.pl>

Co to jest Parrot?

- Wirtualna maszyna (VM)
- Zaprojektowana specjalnie dla Perla 6
- Obsługa innych języków dynamicznych

Perl 6

- „Community rewrite of Perl”
- „Merely the prototype for Perl 7”

- Larry Wall

- Perl should stay Perl

Krótka historia

- Perl conference 4 (rok 2000)
- Primaaprilis
- RFC
- Apokalipsy (apokalypses)
- Egzegezy, objaśnienia (exegesis)
- Streszczenia (synopsis)

Dzień dzisiejszy

- Perl 5 ma się dobrze (Perl6::)
- Perl 6 to vaporware
- Parrot już nie jest vaporware
- Ponie (Perl on New Internal Engine)

Ludzie

- Larry Wall
- Damian Conway
- Dan Sugalski
- Leopold Tötsch

Perl should stay Perl

- Przebudowa zamiast trzęsienia ziemi
- Bez wciskania obiektowości
- Tylko elementy języków funkcyjnych

Zasadnicze zmiany

- Opcjonalny system typów
- Zmiana znaczenia pierwszego znaku przy odwołaniu do zmiennych
- Przebudowa systemu wyrażeń regularnych
- ...
- Większe lub mniejsze zmiany praktycznie w całym języku.

Najbardziej widoczne

- Pierwszy znak zmiennej

```
$tablica[2]    -> @tablica[2]
$hash{klucz}   -> %hash{klucz}
```

- Dereferencja przy indeksowaniu

```
$ref = \@array;
$ref->[2];              # Perl 5
$ref[2]; $ref.[2]; # Perl 6
```

Najbardziej widoczne (2)

- Operator wywołania metody - '.'

```
$obj->metoda();          # Perl 5
$obj.metoda();           # Perl 6
```

- Operator łączenia łańcuchów - '~'

```
$b = $a . '_b';          # Perl 5
$b = $a ~ '_b';          # Perl 6
```

Najbardziej widoczne (3)

- System typów (**opcjonalny!**)

```
my Int $licznik;
my @container is Array of Num;
```

- Prototypy procedur

```
sub FindCat(Str $name) returns Cat {
```

Najbardziej widoczne (4)

- Nawiasy? Jakie nawiasy?

```
if ( $zmienna) { ... }  # Perl 5
if $zmienna { ... }   # Perl 6
```

- Podobnie w warunkach pętli itp.

```
while $zmienna { ... }
```

Operatory

Super operator porównania (DWIM)

```
%hash ~~ $str        $_ ~~ $num       @arr ~~ @arr2
$num ~~ @arr         $_ ~~ $str       $str ~~ /^a.*b/
%hash ~~ /rule/      %hash ~~ @arr    $obj ~~ Dog
2 ~~ @arr            2 ~~ *@arr       %hash ~~ %hash2
```

Nowe operatory

- `:=` # binding
- `? ! + - ~ $(...) @(...) %(...)`
- `xx` - 'a' xx 3 ~~ ('a', 'a', 'a')
- ... #2 semi-infinite list

  #2 class Klasa { ... }
- `//` # $val = function() // $two;

Nowe operatory (2)

- Hiperoperatory

```
np. >>+<<

@res = @arr >>+<< 5

@res = @arr >>*<< @arr2

@positions »++
```

Łączenie operatorów porównujących

```
if 5 > $a > 2 {
```

Nowe operatory (3)

- Połączenia (junctions)

```
if $num == 1|2  {
my $jun = 2 ^ 3;
if $val == $jun {
```

- Pomocnicze

```
all() - &          any() - |
one() - ^          none()
```

Nowe operatory (4)

- `¥ - zip(@arr, @arr2)`

```
# równoległe przetwarzanie tablic
```

- Splat – `*, **`
- `==> i <==`

```
@tab ==> grep {/^\d+/} ==> map {$_ * 2} ==> @wynik;
# UNIX pipes
```

Składnia

- Pętle: for, while, loop, ~~foreach~~, ~~do~~

```
for @arr -> $scalar is rw {...}
while ($line = <$*IN>) {...}
loop { ...; last if ...}
loop $i = 0; $i < 10; $i++ {...}
for (0..Inf) -> $counter {...}
for <$*IN> -> $line {...}
for @array {...}
for @array -> $_ is rw {...}
# $_ jest zawsze rw; parametry nazwane już nie
```

Składnia (2)

- Switch!

```
given $data {
    when /^\d+$/ { say 'Liczba '; continue }
    when /[02468]$/ {say 'Parzysta'}
    when Dog { say 'Szczekajacy Pies'}
    default {say 'Coś innego'}
}
```

Podprocedury

```
my RETTYPE sub NAME ( PARAMS ) TRAITS {...}
sub numcmp ($x, $y) { return $x <=> $y }
$comparison = numcmp(2,7);
$comparison = numcmp(x=>2, y=>7);
$comparison = numcmp(y=>7, x=>2);
$comparison = numcmp(:x(2), :y(7));
```

Podprocedury (2)

- parametry opcjonalne

```
sub my_substr ($str, ?$from = 0, ?$len = Inf) {...}
```

- parametry “nazwane”

```
sub formalize($text, +$case, +$justify) {...}
$formal = formalize($title, case=>'upper');
$formal = formalize($title,justify=>'left');
$formal = formalize($title,
                    :justify«right»,
                    :case«title»);
```

Podprocedury (3)

```
sub foo($x, $y, $z) {...}
@onetothree = 1..3;

foo(1,2,3);  # OK:  3 argumenty
foo(@onetothree); # błąd: tylko 1 arg
foo(*@onetothree); # OK:  @onetothree spłaszczony
  (flattened) do 3 args
```

Typy

low-level storage types

```
bit, int, str, num, ref, bool,
```

high-level object types

```
Bit, Int, Str, Num, Ref, Bool, Array, Hash, IO, Code,
  Routine, Sub, Method, Macro, Rule, Block, Object,
  Grammar, ...
```

Typy (2)

- Typy wartości

```
my Dog $spot; my int @array;
my $spot returns $dog;
my $spot of Dog;
my Rat %ship;
```

- Typy zmiennych

```
my $spot is Scalar;
my $spot is PersistentScalar;
my %book is Hash of Array of Recipe;
```

Traits (cechy?)

- Zmiennych

```
my $pi is constant = 3;
```

- Procedur

```
sub fib is inline {...}
```

- Parametrów

```
is constant, is rw, is ref, is copy
```

Obiekty

```
class Point {
        has $.x;
        has $.y is rw;
        method clear () { $.x = 0; $.y = 0; }
}
my $point = Point.new(x => 2, y => 3);
$a = $point.y;
$point.y = 42;
```

Obiekty (2)

```
class Human does Hitchhiker {
        has $:private;
        method :priv_met () { ... }
}
role Hitchhiker {...}
  # role to specjalna klasa
class Point3d is Point {
       ...
}
```

Reguły

```
$zmienna ~~ s/\w+/słowo/;
$zmienna ~~ s:globally/\w+/słowo/;

rule digits {\d+};
$string ~~ /<digits>/;
```

Reguły (2)

- zbiór nazwanych reguł

```
grammar Gramatyka {
    rule imie {Mariola|Jolanta|
   Maria}
    rule cyfry {\d+}
}
```

- gramatyki mogą po sobie dziedziczyć:

```
grammar MojaGramtyka is Gramatyka { ... }
```

Reguły (3)

- Modyfikatory na początku
- Domyślna „niewrażliwość” na białe znaki
- () zachowuje znaczenie
- [] oznacza (?: ...) z Perla 5 (tj. “non-capturing group”)
- {} oznacza interpretację kodu wewnątrz
- <> ma wiele funkcji

Reguły (4)

```
^   $   ^^   $$
*   *?   +?   ?   ??
<3> <3..6> <1...>
**{3} **{3..5} **{1...} **{3}?
$rule = rule {...}
<[...]> <”...”> <@array> <$rule> <rule>
```

Reguły (5)

- Modyfikatory

```
m:i/test/; m/aaa [:i cent[er|re]]/;

m:ignorecase/test/

m:7th/test/; m:2nd/test/
# globally - od końca poprzedniego dopasowania
# ignorecase
# overlap - od następnego znaku po dopasowaniu
# words - spacje traktowane jako \s+
# cont  - kontynuuj od poprzedniego dopasowania
# perl5
# ...
```

- Wbudowane reguły

```
<ws> <alpha> <digit> <before ...> <after ...>
```

- Perl6::Rules – można wypróbować juz teraz !!

Zakończenie

Przykłady zaczerpnięte zostały z książki “Perl6 and Parrot essentials” oraz z różnych Apokalips, Egzegez i Streszczeń.

http://www.parrotcode.org/

http://dev.perl.org/perl6/

http://www.perl6.org/

http://www.poniecode.org/

dokumentacja do Perl6::

Pytania?

1. Kiedy?

   The current plan is to have a working 6.0.0 compiler ready in the 3rd quarter of 2005.

...

za http://dev.perl.org/perl6/faq.html