Perl 6 in Kontext

Bisher:

Allgemein

Array / Hash

Regex

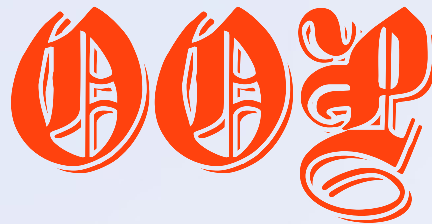
OOP

lichtkind.de

Allgemein

Array / Hash

Regex



Alle großen Themen

Übersicht

Variablen

Parser

Abstraktion

Alle großen Themen

Variablen

Parser

Abstraktion

Ausdrücke

OOP ==> OOL

Array / Hash

Regex

OOP

Operatoren

OOP ==> OOL

Operator

Oriented

Language

Perl 6 in Kontext

Random Thoughts

- The word "apocalypse" historically meant merely "a revealing", and we're using it in that unexciting sense.
- If you ask for RFCs from the general public, you get a lot of interesting but contradictory ideas, because people tend to stake out polar positions, and none of the ideas can build on each other.
- Larry's First Law of Language Redesign: Everyone wants the colon.
- RFCs are rated on "PSA": whether they point out a real Problem, whether they present a viable Solution, and whether that solution is likely to be Accepted as part of Perl 6.
- Languages should be redesigned in roughly the same order as you would present the language to a new user.
- Perl 6 should be malleable enough that it can evolve into the imaginary perfect language, Perl 7. This darwinian imperative implies support for multiple syntaxes above and multiple platforms below.
- Many details may change, but the essence of Perl will remain unchanged. Perl will continue to be a multiparadigmatic, context-sensitive language. We are not turning Perl into any other existing language.
- Migration is important. A Perl 6 interpreter, if invoked as "`perl`", will assume that it is being fed Perl 5 code unless the code starts with a "class" or "module" keyword, or you specifically tell it you're running Perl 6 code in some other way, such as by:

1 Wort zu Operatoren

Operatoren

Piktogramme

Operatoren

Piktogramme
schnelle Orientierung

Operatoren

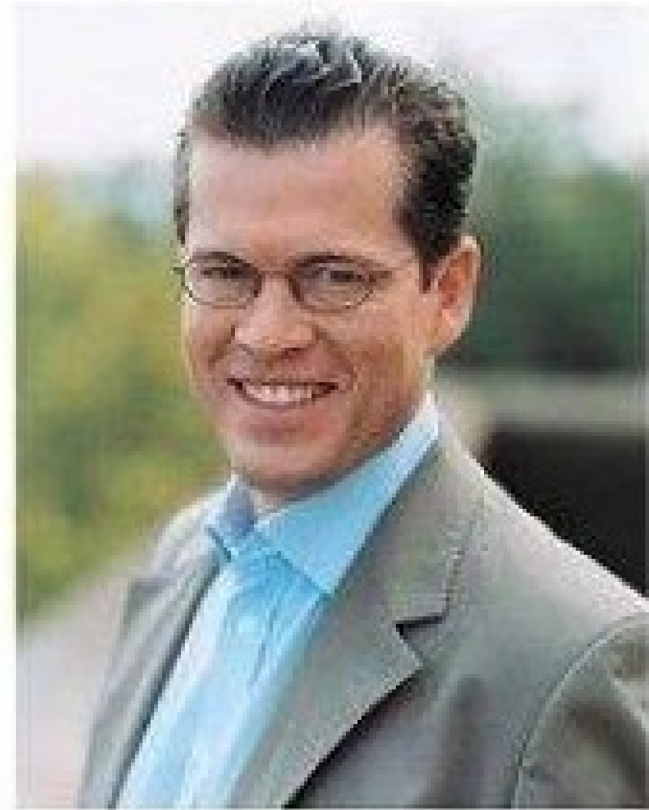
Piktogramme
schnelle Orientierung
wie Einrückungen

Operatoren

Piktogramme
schnelle Orientierung
wie Leerzeilen

Verwechslungsgefahr

Dr. No



No Dr.

Perl 5 in Kontext

Perl 5 in Kontext

wantarray

Kontext: **wantarray**

true (else) - array
false (0|'') - scalar
undef - void

Perl 6 in Kontext

kein wantarray !!!

P6 Innereien

Kontext

=

Datentyp

=

Klasse

Typklassen:

Num

Str

Array

Hash

...

Wie Gewohnt

```
my $l = 12;
my $d = 'text';
```

Aber Implizit:

```
my Num $l = 12;
my Str $d = 'text';
```

Wer Java kennt:

public method to_string {

In P6 meist implizit

```
$var.to_string();
```

Perl 5 in Kontext

```
$nr =()= $str =~ /.../g;
```

Goatse Secret Op

```
$nr =()= $str =~ /.../g;
```

Fordert List Kontext

```
$nr =()= $str =~ /.../g;
```

Explizit in Perl 6

@() array

Explizit in Perl 6

$()	scalar
@()	array
%()	hash
&()	code

Perl 6 in Kontext

$ scalar
@ array
% hash

Invariante Sigils

$ scalar

@ array

% hash

Invariante Sigils

$scalar
@array
%hash

Zeigen kein Kontext

```
$scalar
@array[5]
%hash{'key'}
```

Schreiben wir so

```
$scalar
@array[5]
%hash<key>
```

Sigils

$	scalar
@	positional
%	assoziativ
&	aufrufbar

Kontextoperatoren

$()	scalar
@()	array
%()	hash
&()	code

Langversion

$()	item()
@()	list()
%()	hash()
&()	code()

Klammern optional

$()	item()
@()	list()
%()	hash()
&()	code()

Item Kontext

$()	item()
@()	list()
%()	hash()
&()	code()

List Kontext

$()	item()
@()	list()
%()	hash()
&()	code()

P5 List Kontext

$() item()
flat()
%() hash()
&() code()

Hash Kontext

$()	item()
@()	list()
%()	hash()
&()	code()

Code Kontext

$()	item()
@()	list()
%()	hash()
&()	code()

Mehr Kontext Ops

~ string

+ numeric

? boolean

Negierende Ops

~ string
+ - numeric
? ! boolean

Beispiel ohne ()

~@list
+@list
?@list

Kontext Ops

```
~@list  @list[0].@list[1]
+@list
?@list
```

Kontext Ops

~@list @list[0].@list[1]

+@list @list.elems

?@list

Kontext Ops

```
~@list  @list[0].@list[1]
+@list  @list.elems
?@list  @list.elems > 0
```

Bool Kontext

Bool Kontext

?   !

?^ ?| ?&

^ | &

//   ^^ || && ff   fff

??   !!

Bool Kontext

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Bool Kontext

Bool Kontext

```
my $var = 45;
say ?$var;
```

Bool Kontext

```
my $var = 45;
say ?$var;
```

True

Bool Kontext

```
my $var = 45;
say ?$var;
```

True

Bool::True in Stringkontext

Bool Kontext

```
my $var = 45;
say !$var;
```

False

Bool::False in Stringkont.

Ist das so?

```
my $var = e;
say so($var);
```

True

Bool::True in Stringkontext

Ist das nicht so?

```
my $var = e;
say not $var;
```

False

Bool::False in Stringkont.

Hohe Präzedenz

```
my $var = 45;
say ?$var + 1;
```

Hohe Präzedenz

```
my $var = 45;
say ?$var + 1;
```

2

True in Numkontext = 1

Niedrige Präzedenz

```
my $var = 45;
say so $var + 1;
```

Niedrige Präzedenz

```
my $var = 45;
say so $var + 1;
```

True

46 im Boolkontext = True

Bool Kontext

```
my $var = 45;
say 1 if $var + 1;
```

1

46 im Boolkontext = True

Bool Kontext

```
my $var = 45;
say 1 if $var + 1;
```

1

If unless while until

Das War einfach!

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

What the Op?

?^ ?| ?&

Forciert Bool Kontext

?^ ?| ?&

Ja, ja logische Ops

?^ ?| ?&

1+2 = ?

?^ ?| ?&

Was könnt das sein?

```
say 0 ?|  'wald';
```

Was könnt das sein?

```
say 0 ?|  'wald';
```

True

False or True = True

Was könnt das sein?

```
say 5 ?^  0.0;
```

Was könnt das sein?

```
say 5 ?^  0.0;
```

True

True xor False = True

Ein Sinn eröffnet sich

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Hmmmmm ?

^ | &

Hmmmmm ?

```
$var = 0 |  'wald';
say $var;
```

schlauer ?

```
$var = 0 |  'wald';
say $var;
```

any(0, wald)

Junctions !

```
$var = 0 |  'wald';
say $var;
```

any(0, wald)

wörtlich: 0 oder 'Wald'

Junctions !

```
0 | 1 | 3 | 7 = any(0,1,3,7);
```

Junctions !

2 ~~ 0 | 1 | 3 | 7

Junctions !

2 ~~ 0 | 1 | 3 | 7

False

Junctions !

```
1 == 0 | 1 | 3
```

Junctions !

1 == 0 | 1 | 3

any(False, True, False)

Junctions !

```
if $val == 0 | 1 | 3 { ...
```

Junctions !

```
if $val == 0 | 1 | 3 { ...
```

True

Junctions !

```
if $val == 0 | 1 | 3 { ...
```

```
any(False, True,
False).to_bool
```

Es lichtet sich

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

! Forciert Kontext

// ^^ || && ff fff

Kurzschluß ODER

```
tu_dies()  ||  tu_das();
```

Short Circuit OR

```
tu_dies()   ||   tu_das();

tu_das() unless  tu_dies();
```

Defined OR

```
dies()   //   das();
```

Defined OR

```
dies()   //   das();
```

```
das() unless defined dies();
```

Kurzschluß UND

```
tu_dies()   &&   tu_das();

tu_das()   if   tu_dies();
```

Kurzschluß XOR

```
tu_dies()   ^^   tu_das();
```

Kurzschluß XOR

```
tu_dies()   ^^   tu_das();

my $var = tu_dies();
if not $var { $var }
   else      { tu_das() }
```

Kein else mit unless

```
tu_dies()   ^^   tu_das();

my $var = tu_dies();
if not $var { $var }
   else       { tu_das() }
```

Alle Kurzschluß

```
tu_dies()  ||  tu_das();
tu_dies()  //  tu_das();
tu_dies()  &&  tu_das();
tu_dies()  ^^  tu_das();
```

Grenzwerte

```
$a min $b
$a max $b
$a minmax $b
```

Grenzwerte

$a min $b
$a max $b
minmax @a

Flipflop

```
anfang()   ff   ende();
anfang()   fff   ende();
```

War .. | .... in Skalar K.

```
while … {
  tu() if anfang() ff ende();
  tu() if  anfang() fff ende();
}
```

Fast am Ziel

?   !

?^ ?| ?&

^ | &

// || &&

??   !!

Ternärer Op

??   !!

Ternärer Op

war früher ? :

??   !!

Ternärer Op

war früher ? :
wertet in Boolkontext aus

?? !!

Ternärer Op

war früher ? :

wertet in Boolkontext aus

return Wert unverändert

?? !!

Soweit Klar?

?   !

?^ ?| ?&

^ | &

// || &&

??   !!

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Weiß Jeder :)

+ - * / % %% **

+^ +| +&

+< +>

Modulo

7 / 3

7/3(2.333) | 2

Modulo

7 % 3

Modulo

7 % 3

7 = 3 * 2 + <u>1</u>

ModMod?

7 %% 3

Teilbar

7 %% 3

False => R 1

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Bitweise Logik

+^ +| +&

Bitweiser Shift

+< +>

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Was Vergessen ?

Was Vergessen ?

++

--

Geordnete Mengen

++ after

- - before

cmp

Geordnete Mengen

cmp:

Less, Same, More

Geordnete Mengen

cmp:

Less, Same, More

-1, 0, 1

Reihenfolge in Kontext

<=>

leg

cmp

Reihenfolge in Kontext

<=>	Num Kontext
leg	Str Kontext
cmp	allgemein

Reihenfolge in Kontext

<	Num Kontext
lt	Str Kontext
before	allgemein

Reihenfolge in Kontext

>	Num Kontext
gt	Str Kontext
after	allgemein

Geordnete Mengen

++   1 after

- - 1 before

Gleichheit in Kontext

==	Num Kontext
eq	Str Kontext
===	allgemein

Gleichheit in Kontext

==	Num Kontext
eq	Str Kontext
eqv	allgemein

Gleichheit in Kontext

===	statische Analyse
eqv	dynamisch
=:=	gebunden

Reihenfolge in Kontext

```
if 2 eqv 2.0 {
```

Typ-, dann Inhaltcheck

```
if 2 eqv 2.0 {
Int()  vs.  Rat()
```

Typ-, dann Inhaltcheck

```
if 2 == 2.0 {
```

Typ-, dann Inhaltcheck

```
if 2 == 2.0 {
True (Num Kontext)
```

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

String Kontext

~

~^ ~| ~&

~~ x

Perlige to_string

~

War mal der .

~

```
say 'kombiniere' ~ 'Watson';
```

String Kontext

~

~^ ~| ~&

~~ x

Zeichenweise Logik

~^ ~| ~&

Zeichenweise Logik

1 +| 2 = 3

'a' ~| 'b' = 'c'

String Kontext

~

~^ ~| ~&

~~ x

String Kontext

~

~^ ~| ~&

~~ x

String Kontext

```
say '-' x 80;
```

String Kontext

~

~^ ~| ~&

~~ x

List Kontext

, ... xx X Z

<<== <== ==> ==>>

Komma Operator

```
@fib = 1, 1, 2, 3, 5;
```

Komma Operator

```
$fib = (1, 1, 2, 3, 5);
```

Komma Operator

```
$fib = (1);
say $fib.WHAT;
Int
```

Komma Operator

```
$fib = (1 ,);
say $fib.WHAT;
```

Komma Operator

```
$fib = (1 ,);
say $fib.WHAT;
Parcel
```

Komma Operator

```
$fib = (1 ,);
say $fib.WHAT;
```

Parameter Liste

Capture Kontext

- benannte Parameter
- positionale Parameter

List Kontext

```
, ... xx X Z
<<== <== ==> ==>>
```

Sequence Operator

```
$d = 1, 2 ... 9;
```

Yadda Operator

```
sub befriede { ... }
```

Yadda Operator

sub befriede { ... }
sub befriede { ??? }
sub befriede { !!! }

Sequence Operator

```
$d = 0, 1 ... 9;
```

kann Range Op nicht!

```
$d = 9, 8 ... 0;
```

kann Range Op nicht!

```
$d = 9 .. 0;
```

Sequence Operator

```
$zp = 1, 2, 4... 256;
```

Sequence Operator

```
$fib = 1, 1, *+* ... *;
```

Was vergessen?

```
$d = 0 .. 9;
```

Was vergessen?

```
say 0 .. 9;
```

Keine Liste?

```
say 0 .. 9;
```

```
0..9
```

Bestimmt Kontext

```
say (0 .. 9);
```

Keine Liste ?

```
say (0 .. 9);
```

0..9

Was ist es denn?

```
say (0 .. 9).WHAT;
```

Range ???

```
say (0 .. 9).WHAT;
```

Range

Range ???

```
say 5 ~~ 0 .. 9;
```

True

So geht Liste

```
say @(0..9).WHAT;
```

List

List - Ausgabe?

```
say @(0 .. 9);
```

0 1 2 3 4 5 6 7 8 9

for macht Listkontext

```
say $_ for 0 .. 9;
```

0 1 2 3 4 5 6 7 8 9

for macht Listkontext

```
say for 0 .. 9;
```

for macht Listkontext

```
.say for 0 .. 9;
```

0 1 2 3 4 5 6 7 8 9

List Kontext

```
,  ...  xx  X  Z
<<== <== ==> ==>>
```

Spielen mit Listen

xx X   Z

xx Operator

xx Operator

```
say 'eins zwo eins zwo';
```

xx Operator

```
say 'eins zwo eins zwo';
say q:words(eins zwo) xx 2;
```

xx Operator

```
say 'eins zwo eins zwo';
say q:words(eins zwo) xx 2;
say q:w(eins zwo) xx 2;
```

xx Operator

```
say 'eins zwo eins zwo';
say q:words(eins zwo) xx 2;
say q:w(eins zwo) xx 2;
say qw(eins zwo) xx 2;
```

xx Operator

```
say 'eins zwo eins zwo';
say q:words(eins zwo) xx 2;
say q:w(eins zwo) xx 2;
say qw(eins zwo) xx 2;
say <eins zwo> xx 2;
```

X Operator

```
say <eins zwo> X <dan rabauke>;
```

Kartesisches Produkt

```
say <eins zwo> X
<dan rabauke>;
eins dan eins rabauke
zwo dan zwo rabauke
```

Paarung erhalten

```
say <eins zwo> X <dan rabauke>;
('eins','dan'),('eins','rabauke'),
('zwo','dan'),('zwo','rabauke')
```

Z Operator

```
say <eins zwo> Z
<dan rabauke>;
```

Reißverschluss

```
say <eins zwo> Z <dan rabauke>;
```

eins dan zwo rabauke

Reißverschluss

```
say <eins zwo> zip <dan rabauke>;
```

eins dan zwo rabauke

Reißverschluss

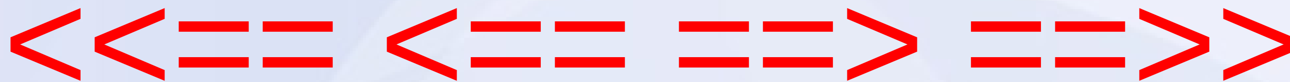
```
for @li Z @re -> $l, $r {
```

Var. nun readwrite

```
for @li Z @re <-> $l,$r {
```

List Kontext

, xx X Z



Schwartz Transform

```
my @output =
  map { $_->[0] }
  sort { $a->[1] cmp $b->[1] }
  map { [$_,expensive_func($_)] }
    @input;
```

Pipe Operator

```
my @output
  <== map { $_[0] }
  <== sort { $^a[1] cmp $^b[1] }
  <== map { [$_,expensive_func($_)] }
  <== @input;
```

Andere Richtung

```
@input
  ==> map { [$_,expensive_func($_)] }
  ==> sort { $^a[1] cmp $^b[1] }
  ==> map { $_[0] }
  ==> my @output;
```

Append Mode

```
my @output
 <<== map { $_[0] }
 <<== sort { $^a[1] cmp $^b[1] }
 <<== map { [$_,expensive_func($_)] }
 <<== @input;
```

Pointy Sub

```
for @input -> $i { ...
```

List Kontext

, xx X Z

<<== <== ==> ==>>

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op =

```
@summe  += 3;
```

Meta Op !

```
if $alter  !< 18 {
```

Meta Op !

```
if $alter  !< 18 {

# echter P6 code
```

Meta Op R

```
$alter  = 2 R- 18;

# == 16
```

Meta Op S

```
$alter  = 2 S- 18;

# == -16
```

Meta Op S

```
$alter  = 2 S- 18;

# kein sichtbarer Effekt
```

Meta Op S

```
$alter  = 2 S- 18;

# ! Parallelisierung
```

Meta Op S

```
$alter  = 2 S- 18;

# ! später wichtiger
```

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op X

Wir erinnern

say <1 2> X <a b>

1 a 1 b 2 a 2 b

Wir erinnern

<1 2> X <a b>

<1 a>,<1 b>,<2 a>,<2 b>

Kartesisches Produkt

<1 2> X <a b>

('1','a'),('1','b'),('2','a'),('2','b')

Kartesischen Paare

<1 2> X~ <a b>

'1a','1b','2a','2b'

aus 'a' wird keine Num

<1 2> X+ <a b>

Stacktrace

Kartesischen Paare

<1 2> X* <3 4>

3, 4, 6, 8

Meta Op Z

manche ahnen was kommt

Metaop Z

<1 2> Z* <3 4>

3, 8

Metaop Z

```
(<1 2>;<3 4>).zipwith(&[*])
```

```
3, 8
```

Metaop Z

```
(<1 2>;<3 4>).zipwith(&[*])
```

```
<1 2> Z* <3 4>
```

Metaop Z

```
(<1 2>;<3 4>).zip()
```

```
<1 2> Z <3 4>
```

Metaop Z

```
(<1 2>;<3 4>).cross()
```

```
<1 2> X <3 4>
```

Metaop Z

```
(<1 2>;<3 4>).crosswith(&[*])
```

```
<1 2> X* <3 4>
```

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op []

Mach mir den Gauss

```
(1..100).reduce(&[+])
```

Forciert List Kontext

```
(1..100).reduce(&[+])

[+] 1 .. 100
```

Forciert List Kontext

True

[<] 1 .. 100

Whats datn?

```
(1..100).triangle(&[+])

[\+]  1 .. 100
```

Whats datn?

1, 3, 6

[\+] 1 .. 3

Whats datn?

1=1, 1+2=3, 1+2+3=6

[\+] 1 .. 3

Meta Op <<

Geburtstagsfeier !!!

```
@alter >>+=>> 1;
```

Alle werden älter

```
@alter == 18, 22, 35;

@alter = @alter >>+>> 1;

@alter == 19, 23, 36;
```

Nur einer wird älter

```
@alter == 18, 22, 35;

@alter = @alter <<+<< 1;

@alter == 19;
```

Interessante Fälle

<18, 22, 35> >>+<< <1, 2>

<18, 22, 35> <<+>> <1, 2>

Interessante Fälle

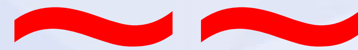
<18, 22, 35> >>+<< <1, 2>

ERROR

<18, 22, 35> <<+>> <1, 2>

19, 24, 36

Komplexität ++



heute nicht

Danke