Perl 6 v Kontextu

doposud:

všeobecný

Array / Hash

Regex

OOP

lichtkind.de

všeobecný

Array / Hash

Regex

OOP

Velke Thema

všeobecný

Proměnne

Parser

Abstrakce

Velke Thema

Proměnne

Parser

Abstrakce

VÝRAZE

OOP ==> OOL

Array / Hash

Regex

OOP

Operator

OOP ==> OOL

Operator

Oriented

Language

Perl 6 v Kontextu

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 Slovo o Operatoru

Operatory

piktogram

Operatory

piktogram (obraz)

schnelle Orientierung

Operatory

piktogram (obraz)

rychla orientace

jake posunuty řádky

Operatory

piktogram (obraz)

rychla orientace

jako prazdne řádky

Verwechslungsgefahr

Dr. No

No Dr.

Perl 5 v Kontextu

Perl 5 v Kontextu

wantarray

Kontext: **wantarray**

true (else) - array

false (0|'') - scalar

undef - void

Perl 6 v Kontextu

žádný wantarray !!!

P6 Vnitřnosti

Kontext

=

Data Typ

=

Třída

Typove Třídy:

Num

Str

Array

Hash

...

Jak Znamo:

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

Ale kdo chce:

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

Kdo Java zna:

public method to_string {

V P6 nejvic implicitni

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

Perl 5 v Kontextu

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

Goatse Tajni Op

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

Žádá Listinovy K.

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

Explicitní v Perl 6

@() array

Explicitní v Perl 6

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

Perl 6 v Kontextu

$ scalar

@ array

% hash

Invariantne Sigile

$ scalar

@ array

% hash

Neměnný Znamení

$scalar
@array
%hash

Neukazujou Kontext

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

Pišeme to ted tak

$scalar

@array[5]

%hash<key>

Sigils

$	scalarni
@	positionalni
%	asociativni
&	vyvolovát.

Kontextoperator

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

Dlouha Verse

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

Závorky Nepovinný

$()	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()

Vice Kontext Op

~ string
+ numeric
? boolean

Negativni Op

~ string

+ - numeric

? ! boolean

Příklad bez ()

~@list
+@list
?@list

Příklad bez ()

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

Příklad bez ()

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

Příklad bez ()

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

Bool Kontext

?

Bool Kontext

? !

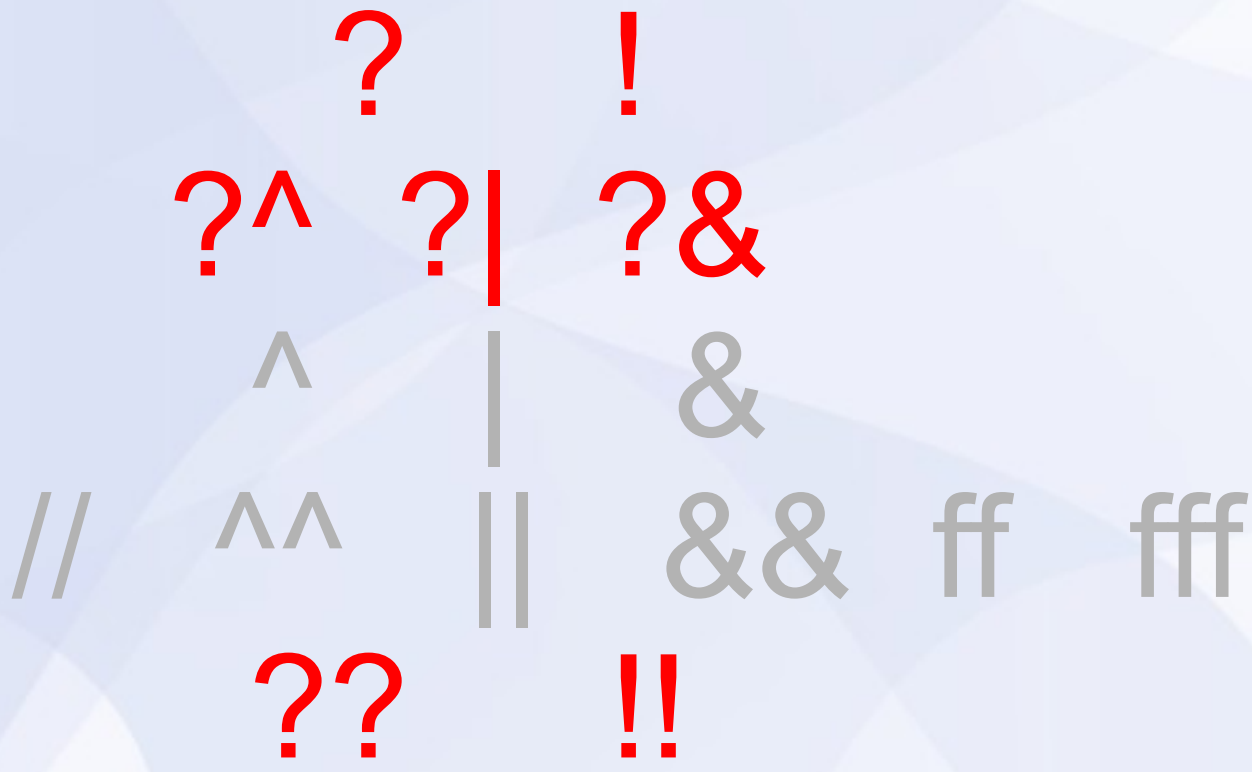
?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Bool Kontext



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 v String Kontext

Bool Kontext

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

False

Bool::False v String Kont.

Je to tak (so)?

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

True

Bool::True v String Kontext

Neni (not) to tak ?

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

False

Bool::False v String Kont.

Vysoka Precedence

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

Vysoka Precedence

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

2

True v Num Kontext = 1

Niska Precedence

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

Niska Precedence

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

True

46 v Bool Kontext = True

Bool Kontext

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

1

46 v Bool Kontext = True

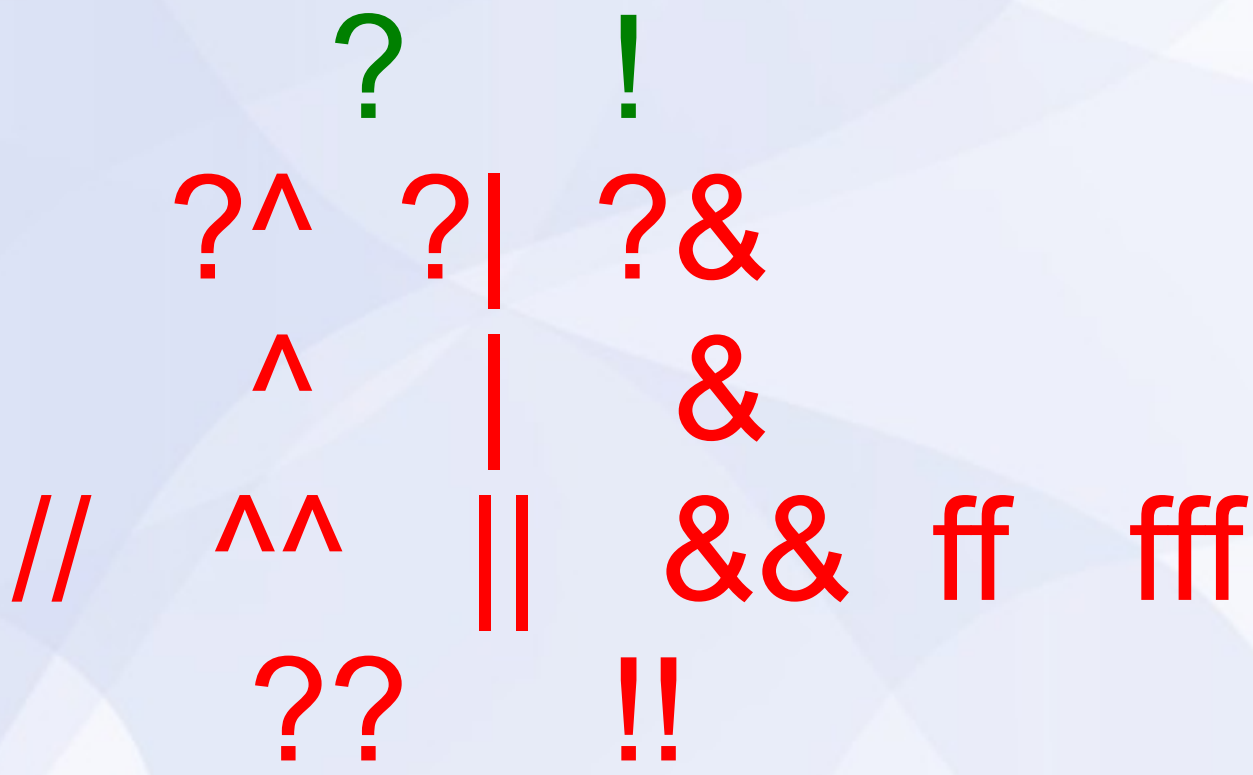
Bool Kontext

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

1

If unless while until

To Bylo Snadne!



Cože?

?^ ?| ?&

Chce Bool Kontext

?^ ?| ?&

No ty logicke Op

?^ ?| ?&

1+2 = ?

?^ ?| ?&

Co to mohlo byt?

```
say 0 ?|  'les';
```

Co to mohlo byt?

```
say 0 ?|  'les';
```

True

False or True = True

Co to mohlo byt?

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

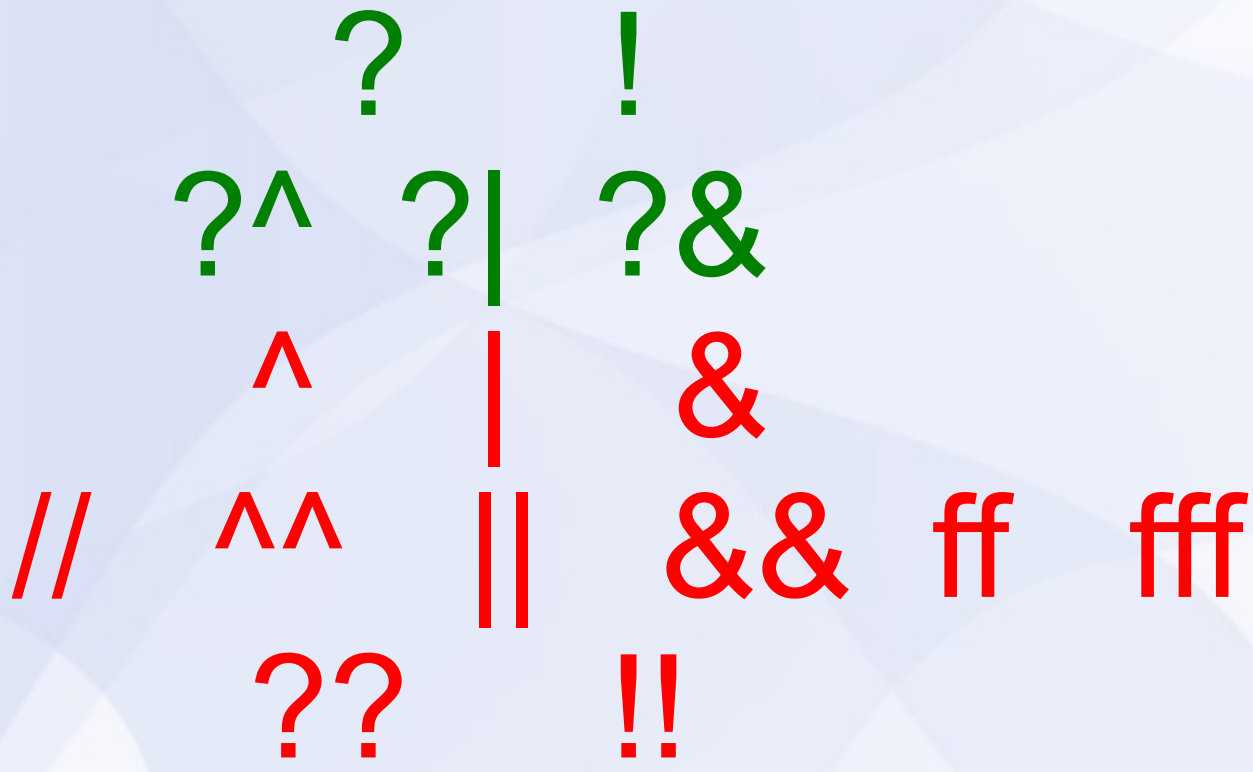
Co to mohlo byt?

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

True

True xor False = True

Otevira se smysl



Hmmmmm ?

^ | &

Hmmmmm ?

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

Chytřejši ?

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

any(0, 'les')

Junctions !

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

any(0, les)

doslova: 0 nebo 'les'

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
```

Rozjasni se

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Nechce Kontext

// ^^ || && ff fff

Zkratkovy NEBO

```
dělej_tak() || dělej_jinak();
```

Short Circuit OR

dělej_tak() || jinak();

jinak() **unless** dělej_tak();

Defined OR

```
tak()   //   jinak();
```

Defined OR

```
tak()  //  jinak();
```

```
jinak() unless defined tak();
```

Zkratkove **A**

tak() && dělej_jinak();

dělej_jinak() **if** tak();

Zkratkovy XOR

tak() ^^ jinak();

eXkluzivni NEBO

```
tak()  ^^  jinak();
```

```
my $var = tak();
if not $var { $var }
   else       { jinak() }
```

Žádný **else** s **unless**

```
tak()  ^^  jinak();
```

```
my $var = tak();
if not $var { $var }
   else       { jinak() }
```

Všechny Zkratky

```
tak()  ||  jinak();
tak()  //  jinak();
tak()  &&  jinak();
tak()  ^^  jinak();
```

Hraniční Hodnoti

$a min $b

$a max $b

$a minmax $b

Hraniční Hodnoti

$a min $b

$a max $b

minmax @a

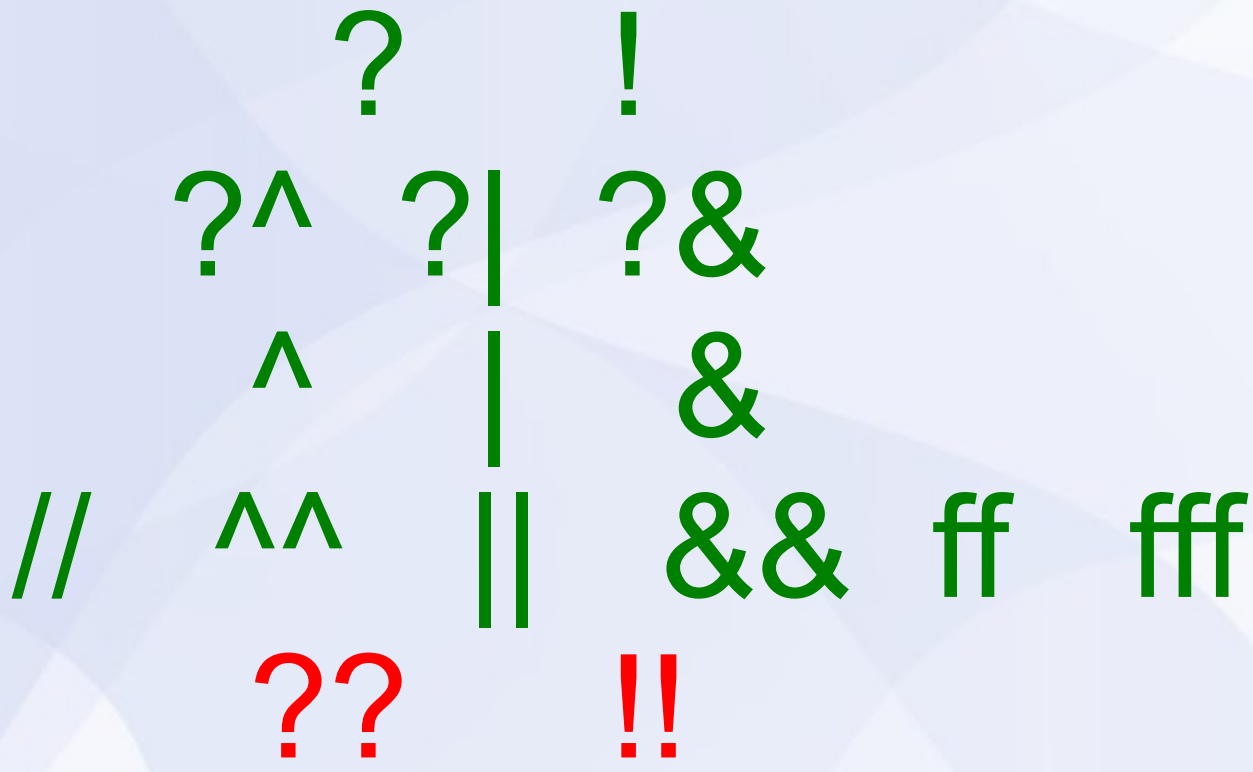
Flipflop

začátek() ff konec();
začátek() fff konec();

War .. | … in Skalar K.

```
while … {
  dělej() if  začátek() ff konec();
  dělej() if  začátek() fff konec();
}
```

Skoro u Cile



Ternärer Op

?? !!

Ternary Op

bylo ? :

?? !!

Ternary Op

bylo ? :
vyhodnoti v Bool Kontext

?? !!

Ternary Op

bylo ? :
vyhodnoti v Bool Kontext
nezměněný hodnoty

?? !!

Všechno jasny?

```
        ?    !
     ?^  ?|  ?&
       ^    |    &
  //   ^^   ||   &&  ff   fff
        ??     !!
```

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Zna každej :)

\+ - * / % %% **

+^ +| +&

+< +>

Modulo

7 / 3

7/3(2.333) | 2

Modulo

7 % 3

Modulo

$$7 \% 3$$

$$7 = 3 * 2 + 1$$

ModMod?

7 %%% 3

Dělitelný

7 %% 3

False => Zbytek 1

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Bit-ova Logika

+^ +| +&

Bit-ova Logika

(bivalo)

^ | &

+^ +| +&

Bit-ovej Shift

+< +>

Bit-ovej Shift

(bivalo)

<< >>

+< +>

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Něco jsem zapomněl?

Něco jsem zapomněl?

++

--

Seřadany Množství

++ after

- - before

cmp

Seřadany Množství

cmp:

Less, Same, More

Seřadany Množství

cmp:

Less, Same, More

-1, 0, 1

Pořadí v Kontextu

<=>

leg

cmp

Pořadí v Kontextu

<=>	Num Kontext
leg	Str Kontext
cmp	všude

Pořadí v Kontextu

<	Num Kontext
lt	Str Kontext
before	všude

Pořadí v Kontextu

>	Num Kontext
gt	Str Kontext
after	všude

Seřadany Množství

++ 1 after

- - 1 before

Rovnost v Kontextu

==	Num Kontext
eq	Str Kontext
===	všeobecný

Rovnost v Kontextu

==	Num Kontext
eq	Str Kontext
eqv	všeobecný

Rovnost v Kontextu

===	staticka Analysa
eqv	dynamicka
=:=	vazany

Pořadí v Kontextu

```
if 2 eqv 2.0 {
```

Datatyp, potom Obsah

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

Datatyp, potom Obsah

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

Datatyp, potom Obsah

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

Numeric Konec

+ - * / % %% **

+^ +| +&

+< +>

String Kontext

~

~^ ~| ~&

~~ x

Perlivy to_string

~

To bylo jednou .

~

```
say 'kombinuj' ~ 'Watson';
```

String Kontext

~

~^ ~| ~&

~~ x

Pismenkova Logika

~^ ~| ~&

Pismenkova Logika

1 +| 2 = 3

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

String Kontext

~

~^ ~| ~&

~~ x

String Kontext

~

~^ ~| ~&

~~ x

String Kontext

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

String Kontext

~

~^ ~| ~&

~~ x

Listinovy Kontext

, ... xx X Z

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

Čárkovy Operator

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

Čárkovy Operator

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

Čárkovy Operator

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

```
Int
```

Čárkovy Operator

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

Čárkovy Operator

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

Parcel

Komma Operator

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

Listina Parametru

Capture Kontext

| pojmenute Parametry
|| pozicionalni Parametry

Listinovy Kontext

, ... xx X Z

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

Sequence Operator

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

Yadda Operator

sub usmířit { … }

Yadda Operator

sub usmířit { … }
sub usmířit { ??? }
sub usmířit { !!! }

Sequence Operator

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

Range Op neumi to!

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

Range Op neumi to!

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

Sequence Operator

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

Sequence Operator

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

Něco jsem zapomněl?

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

Něco jsem zapomněl?

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

Žádna Listina?

say 0 .. 9;

0..9

Urči Kontext

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

Žádna Listina ?

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

```
0..9
```

Co proboha je to?

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

Range ???

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

Range

Range ???

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

True

Tak se děla Listina

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

List

List - Výdej?

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

```
0 1 2 3 4 5 6 7 8 9
```

for chce Listkontext

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

```
0 1 2 3 4 5 6 7 8 9
```

for chce Listkontext

say for 0 .. 9;

for chce Listkontext

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

```
0 1 2 3 4 5 6 7 8 9
```

Listinovy Kontext

, ... xx X Z

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

Hry s Listinamy

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>;

Kartézsky Produkt

say <eins zwo> X <dan rabauke>;

eins dan eins rabauke

zwo dan zwo rabauke

Dostat Pary

say <eins zwo> X <dan rabauke>;

('eins','dan'),('eins','rabauke'),
('zwo','dan'),('zwo','rabauke')

Z Operator

say <eins zwo> Z <dan rabauke>;

Zip

say <eins zwo> Z <dan rabauke>;

eins dan zwo rabauke

Zip

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

```
eins dan zwo rabauke
```

Zip jako Op

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

Proměnná teď rw

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

Listinovy Kontext

, xx X Z

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

Schwartz Transform

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

Pipe Operator

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

Druhý Směr

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

Append Mode

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

Pointy Sub

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

List Kontext

, xx X Z

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

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op =

@suma += 3;

Meta Op !

```
if $věk  !< 18 {
```

Meta Op !

```
if $věk  !< 18 {

# pravej P6 kod
```

Meta Op R

```
$věk  = 2 R- 18;

# == 16
```

Meta Op S

```
$věk  = 2 S- 18;

# == -16
```

Meta Op S

```
$věk  = 2 S- 18;

# nic nevidim
```

Meta Op S

```
$věk  = 2 S- 18;

# ne paralelni !!!
```

Meta Op S

```
$věk  = 2 S- 18;

# ! později důležitý
```

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op X

My Vzpomenem Se

say <1 2> X <a b>

1 a 1 b 2 a 2 b

My Vzpomenem Se

<1 2> X <a b>

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

Kartézsky Produkt

<1 2> X <a b>

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

Kartézske Pary

<1 2> X~ <a b>

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

s 'a' nebude číslo

<1 2> X+ <a b>

Stacktrace

Kartézske Pary

<1 2> X* <3 4>

3, 4, 6, 8

Meta Op Z

některý tuši co přijde

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

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

<1 2> Z <3 4>

Metaop

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

<1 2> X <3 4>

Metaop

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

<1 2> X* <3 4>

MetaOps

= !

X Z R S

[] [\]

<< >>

Meta Op []

Dělej jak Gauss

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

žáda kontext listiny

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

```
[+] 1 .. 100
```

žáda kontext listiny

True

[<] 1 .. 100

Co je to?

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

```
[\+]  1 .. 100
```

Co je to?

1, 3, 6

[\+] 1 .. 3

Co je to?

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

[\+] 1 .. 3

Hyper Op <<

Narozeniny !!!

@věk >>+=>> 1;

Všichni starnou

@věkr == 18, 22, 35;

@věk = @věk >>+>> 1;

@věk == 19, 23, 36;

Jenom jeden starne

```
@věk == 18, 22, 35;

@věk = @věk <<+<< 1;

@věk == 19;
```

Zajimave Pady

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

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

Zajimave Pady

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

ERROR

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

19, 24, 36

komplexnost ++

~~

Ne dnes

Děkuji