Perl 6 in context

Up to now:

`general`

Array / Hash

Regex

OOP

lichtkind.de

general

Array / Hash

Regex

OOP

big topics

general

variable

parser

abstraction

big topics

variable

parser

abstraction

expression

OOP ==> OOL

Array / Hash

Regex

OOP

Operator

OOP ==> OOL

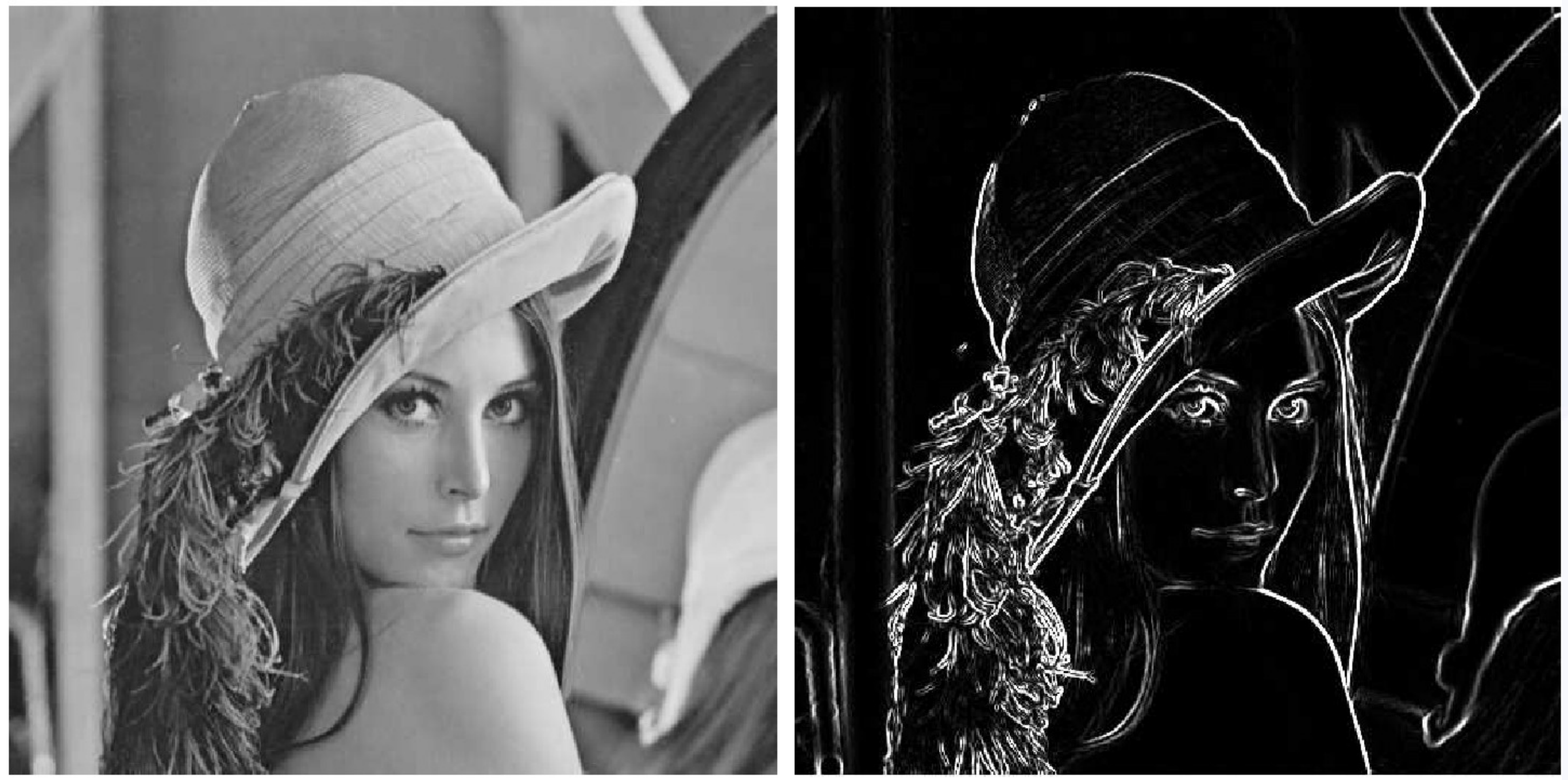
Operator
Oriented
Language

word about operators

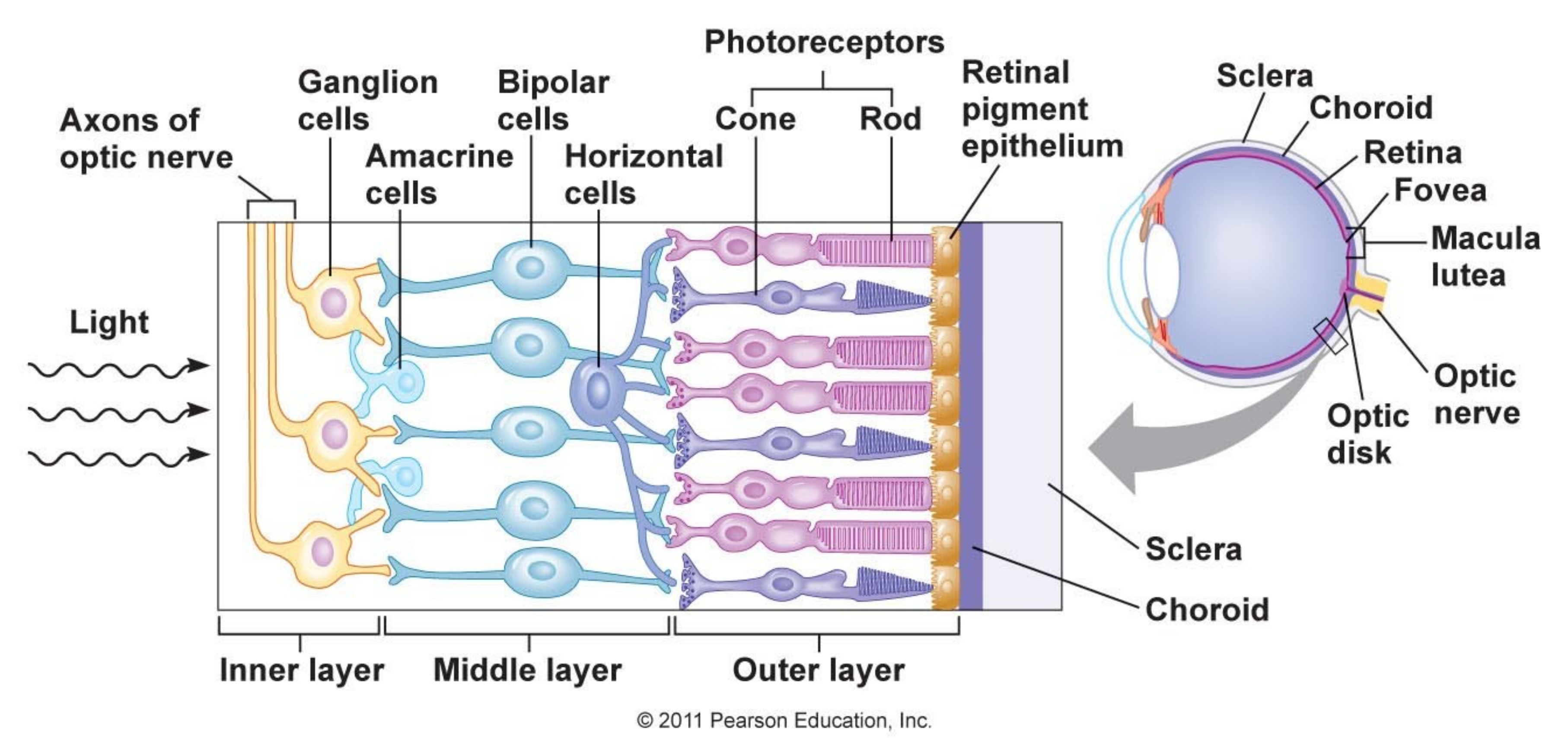
operator

pictogram

Edge Detection

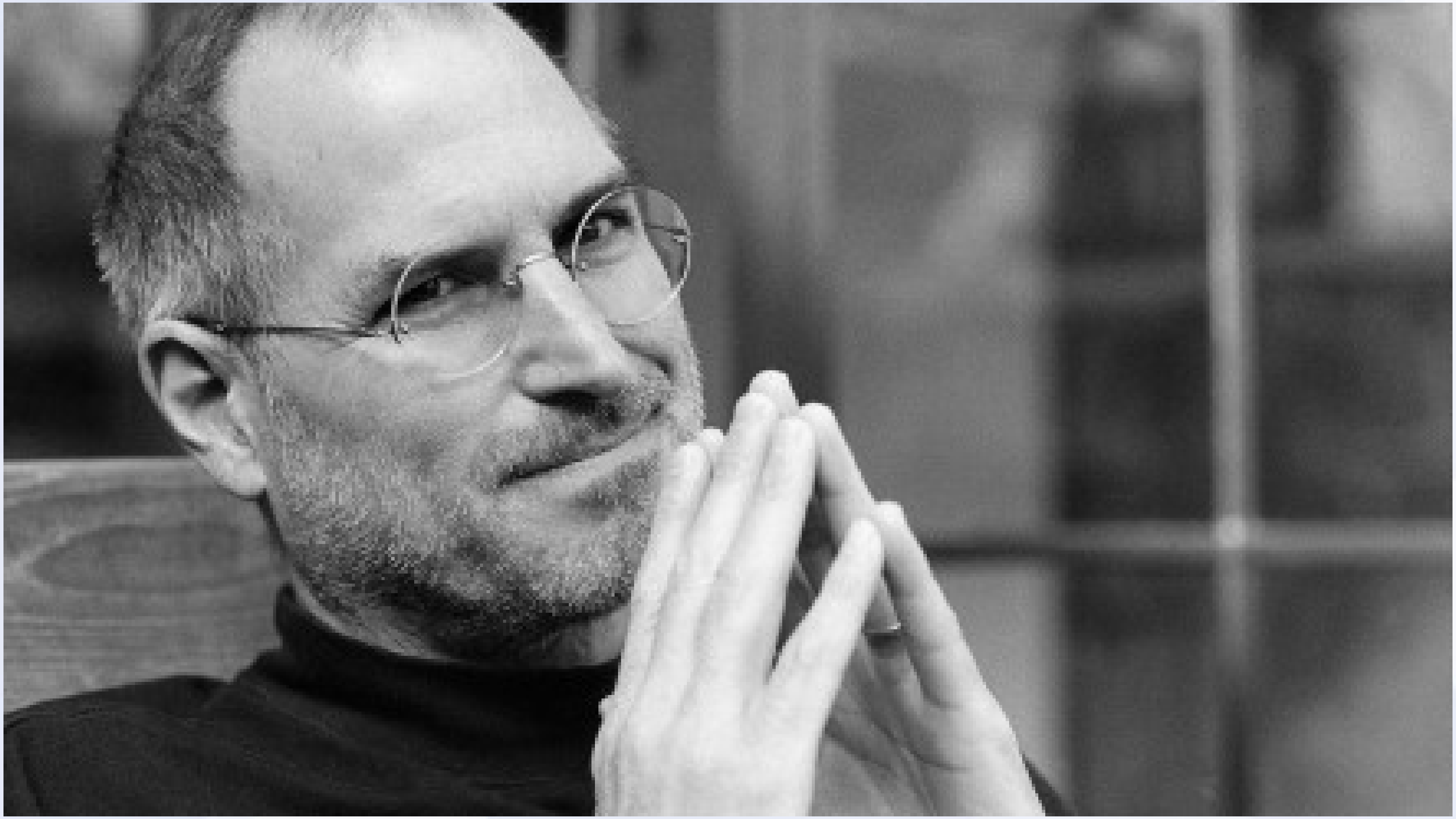


Done By Retina

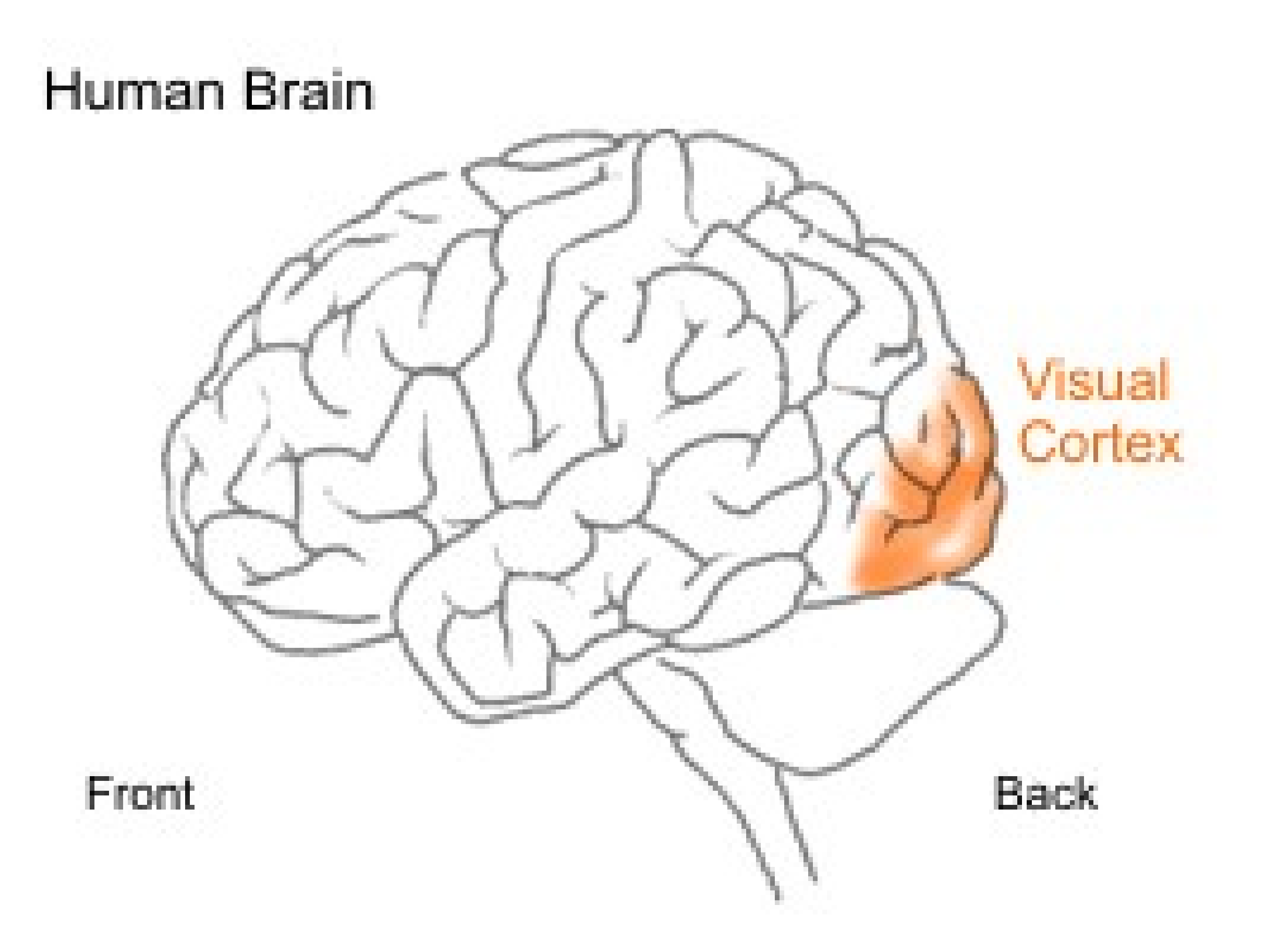


© 2011 Pearson Education, Inc.

I forgot to mention



Visual Cortex



information / min

incoming:
12 000 000

conscious: 40

operator

pictogram
quick orientation

Visual Recognition

pictogram
quick orientation
like empty line

Visual Recognition

pictogram

quick orientation

like indentation

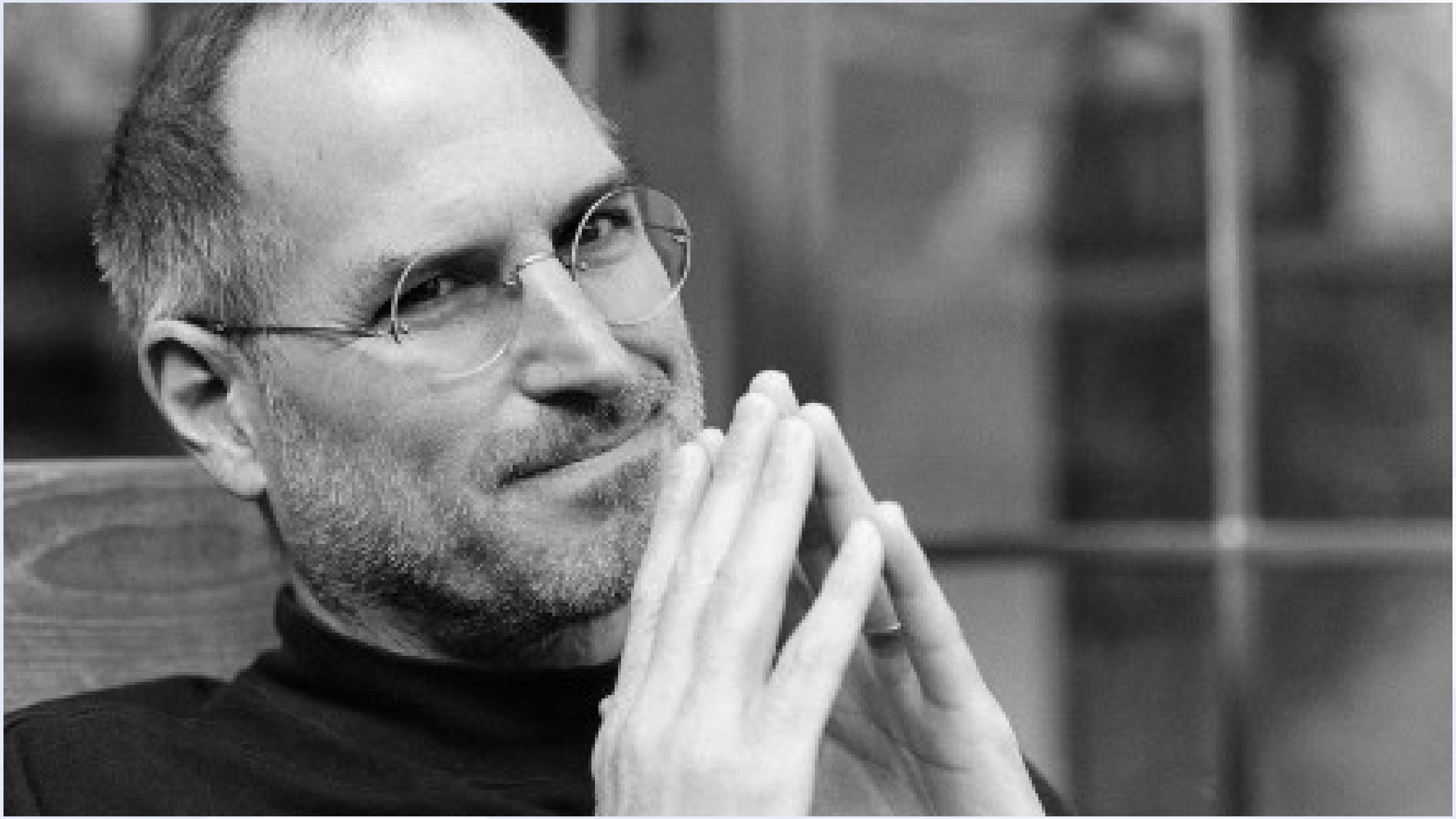
Visual Recognition

pictogram

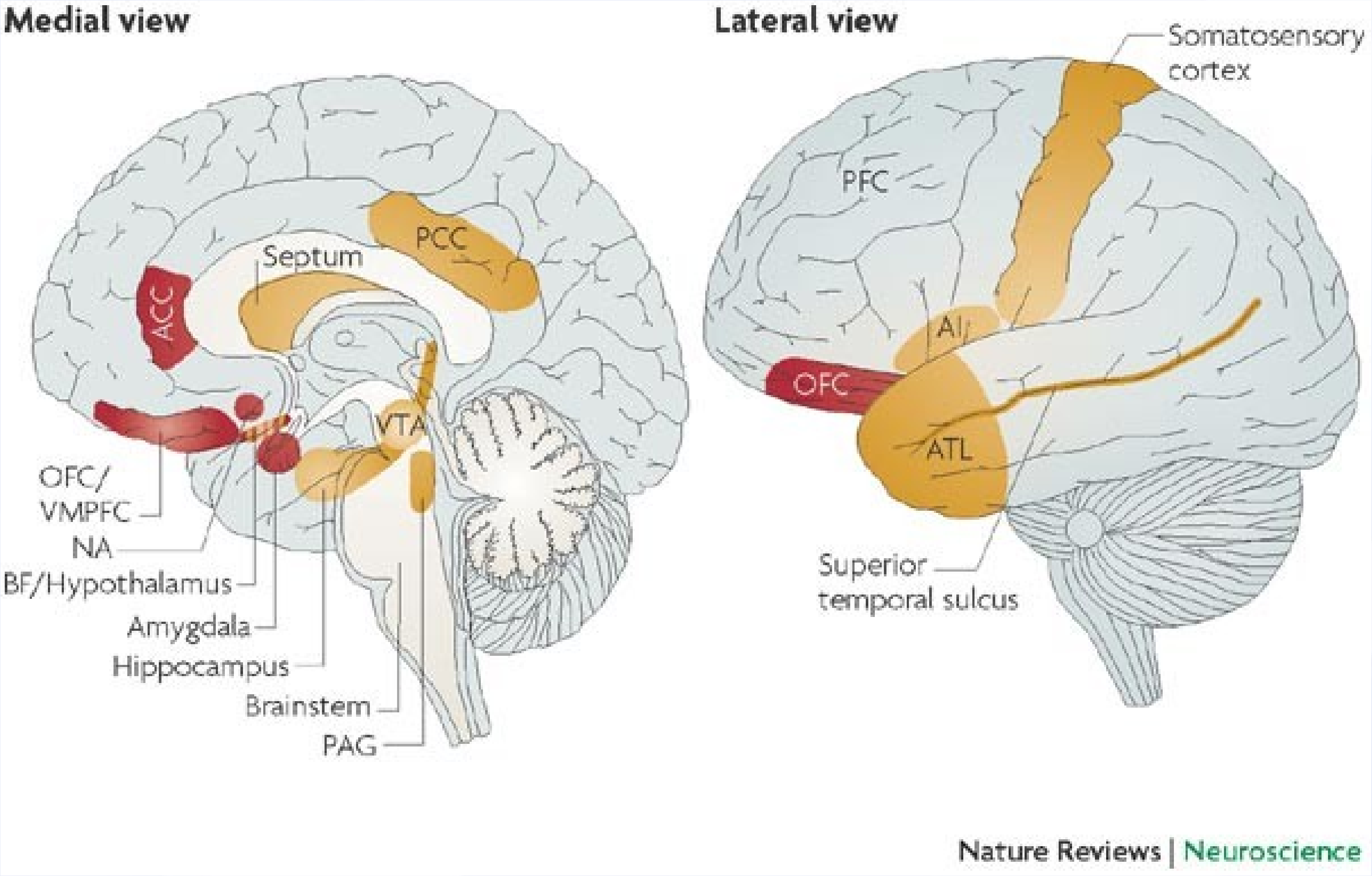
quick orientation

decor. comments

One more thing ...



emotional brain



Antonio Damasio

Antonio Damasio

Leading Neurologist

Emotion != Feelings

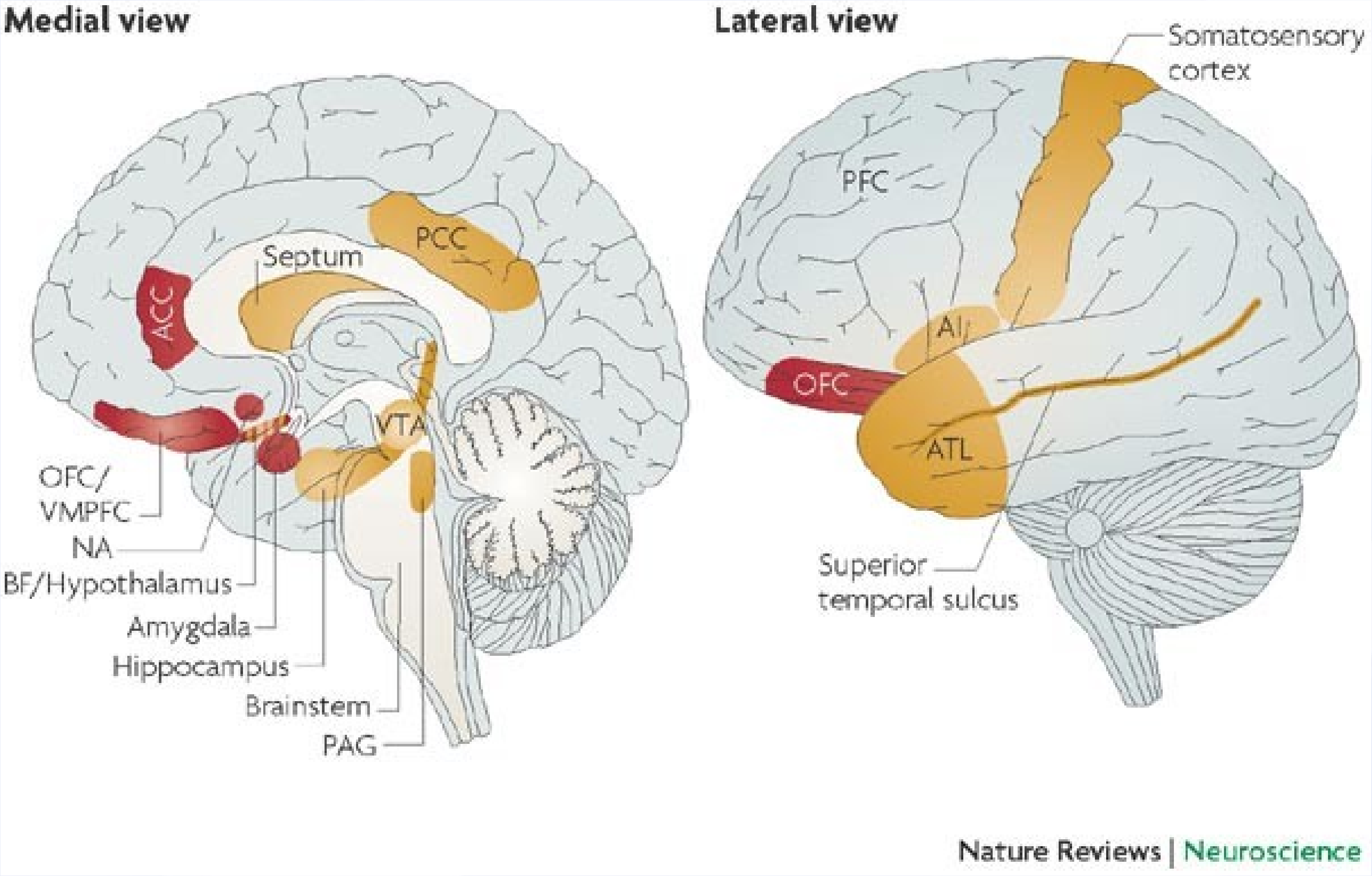
Feeling

Emotion

Damasio:

Emotion is how brain deal with huge data quantities.

does association



Nature Reviews | Neuroscience

Emotions:

triggered (words|pic)
values & judgement
enables memory

TIMTOWTDI =

Respect your emotional wiring (experience)

TIMTOWTDI =

less emotional stress = higher productivity

Ambiguous:

Java: Str + Str

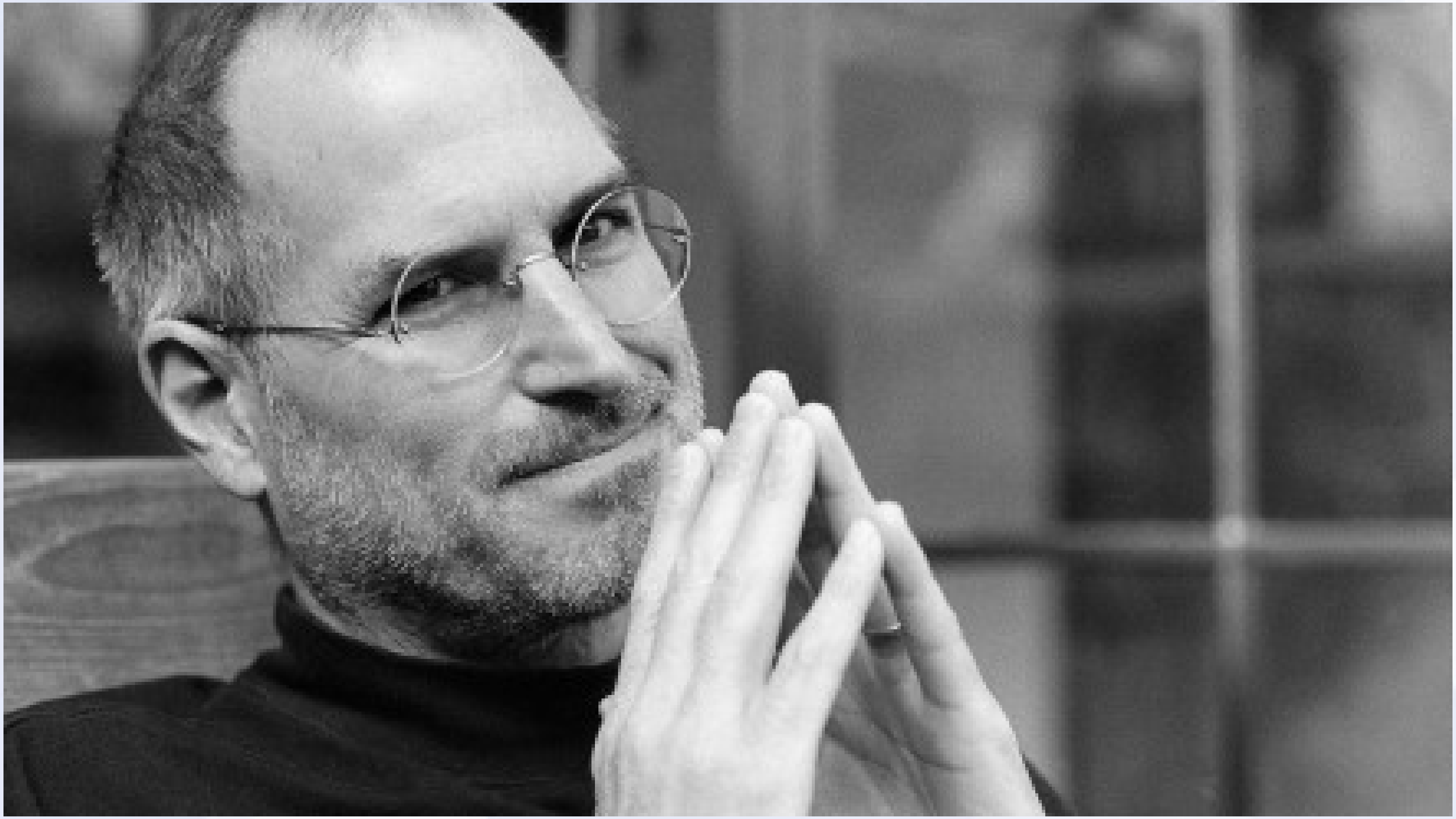
Not Ambiguous:

Perl 5/6: Num + Num

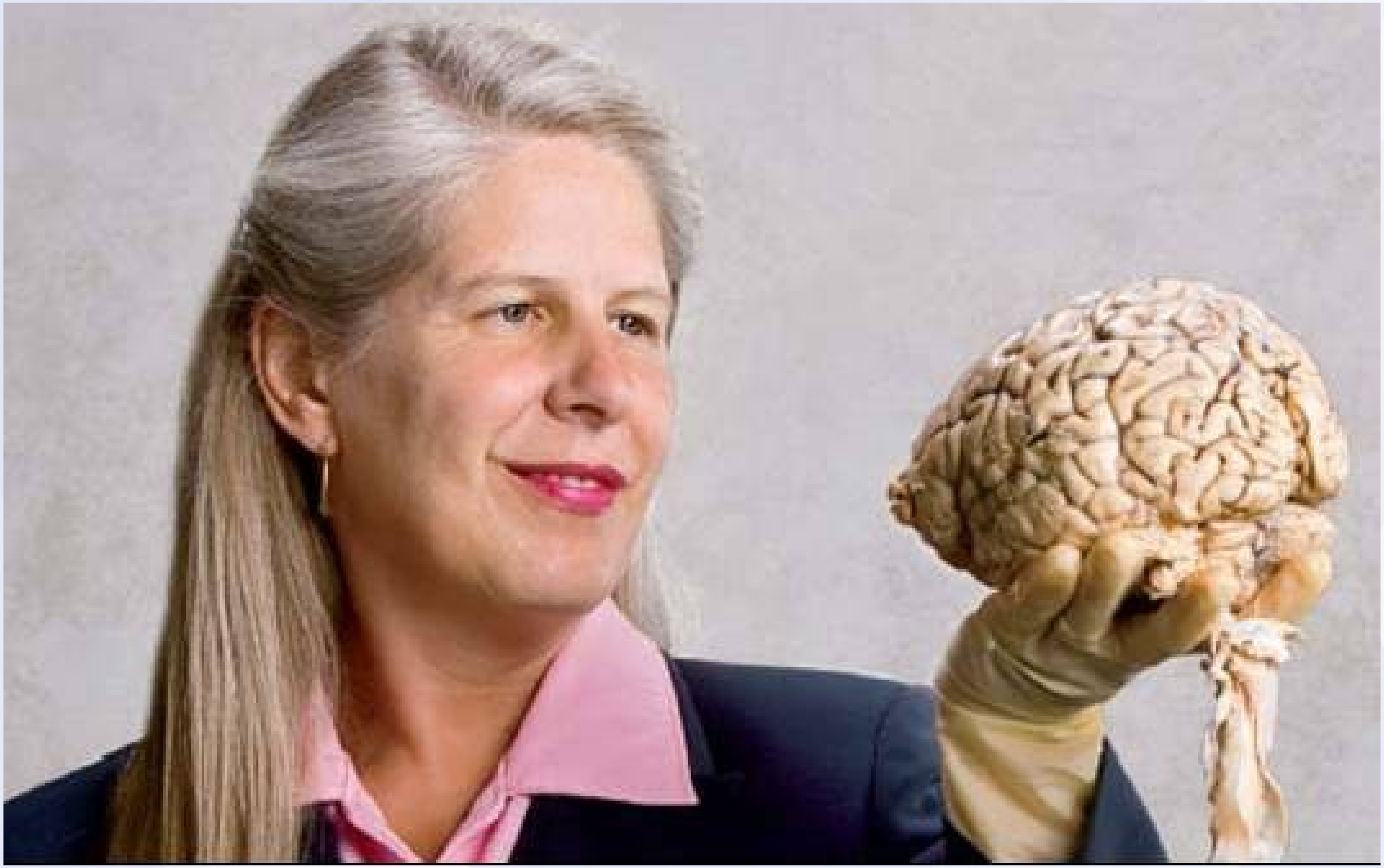
Topicalizer

Perl 6:
class instead
package

One more thing ...



Jill Bolte



Jill Bolte:

Neuroanatomist having left side stroke, experiencing just the emotional mind

Jill Bolte:

left brain works in sequence (future/past) and enables language

Jill Bolte:

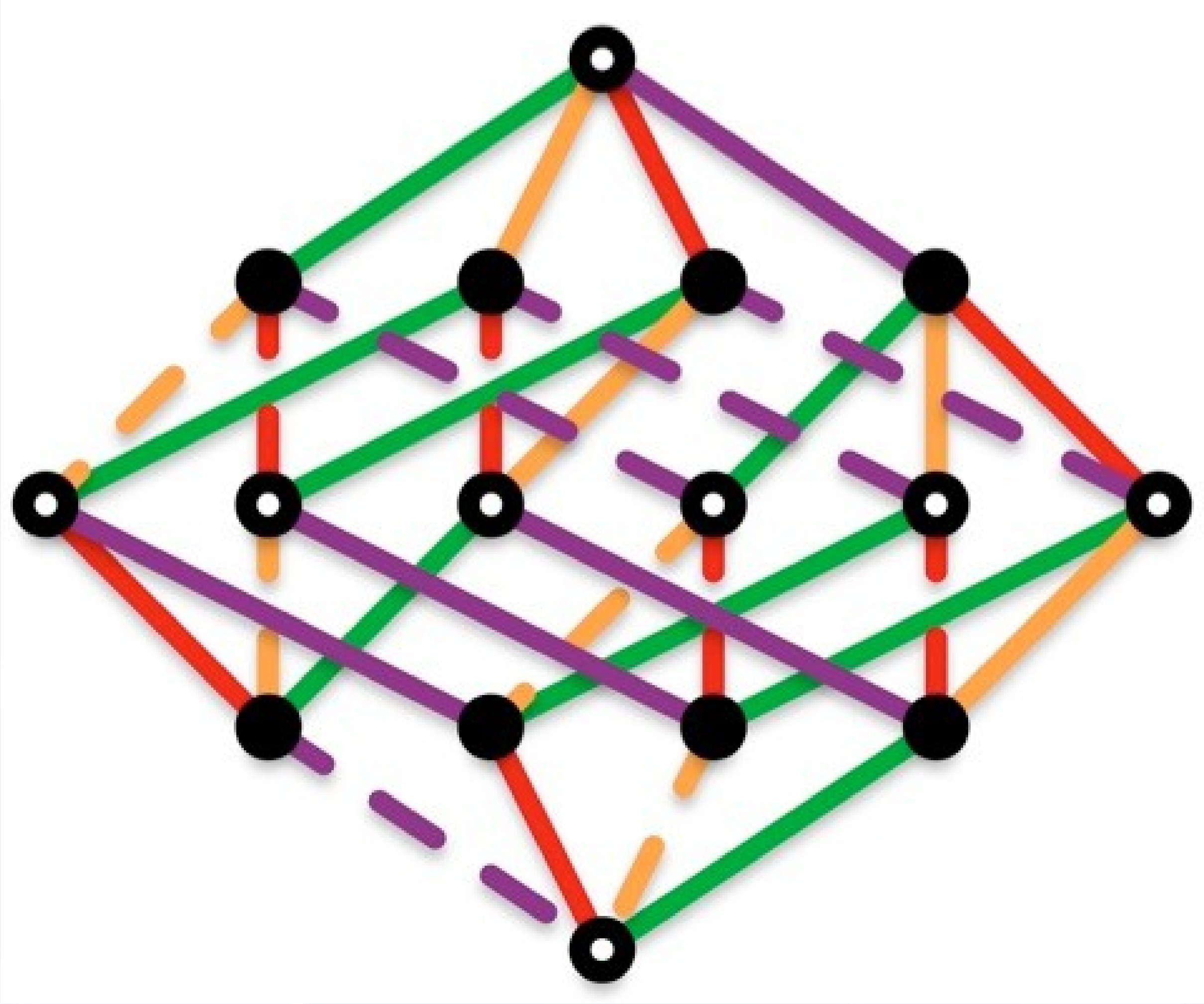
right brain works in parallel, cares about now, emotions, whole picture, graphics

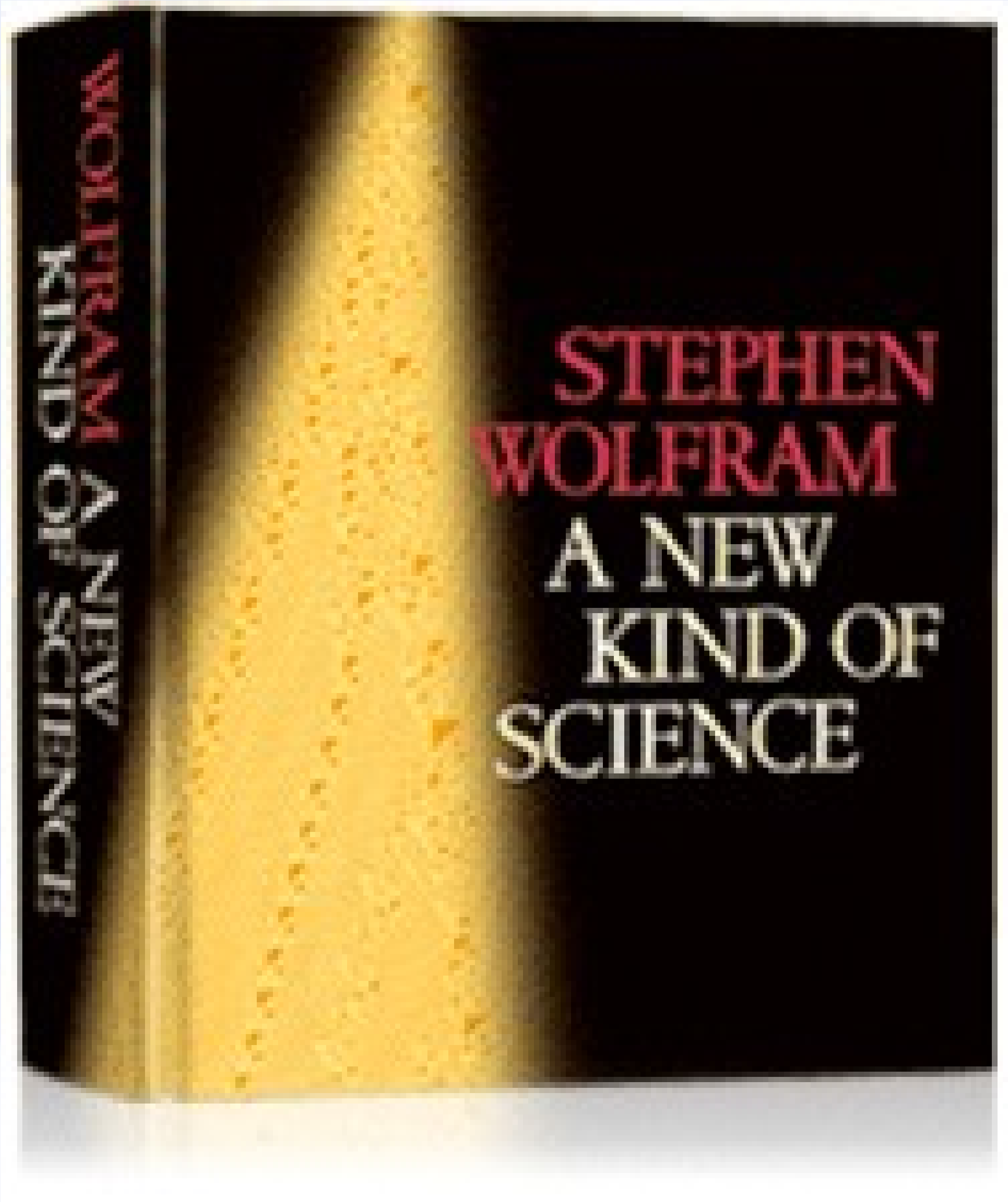
conclusio:

use right brain to grok complex systems

James Gates

Adinkra:



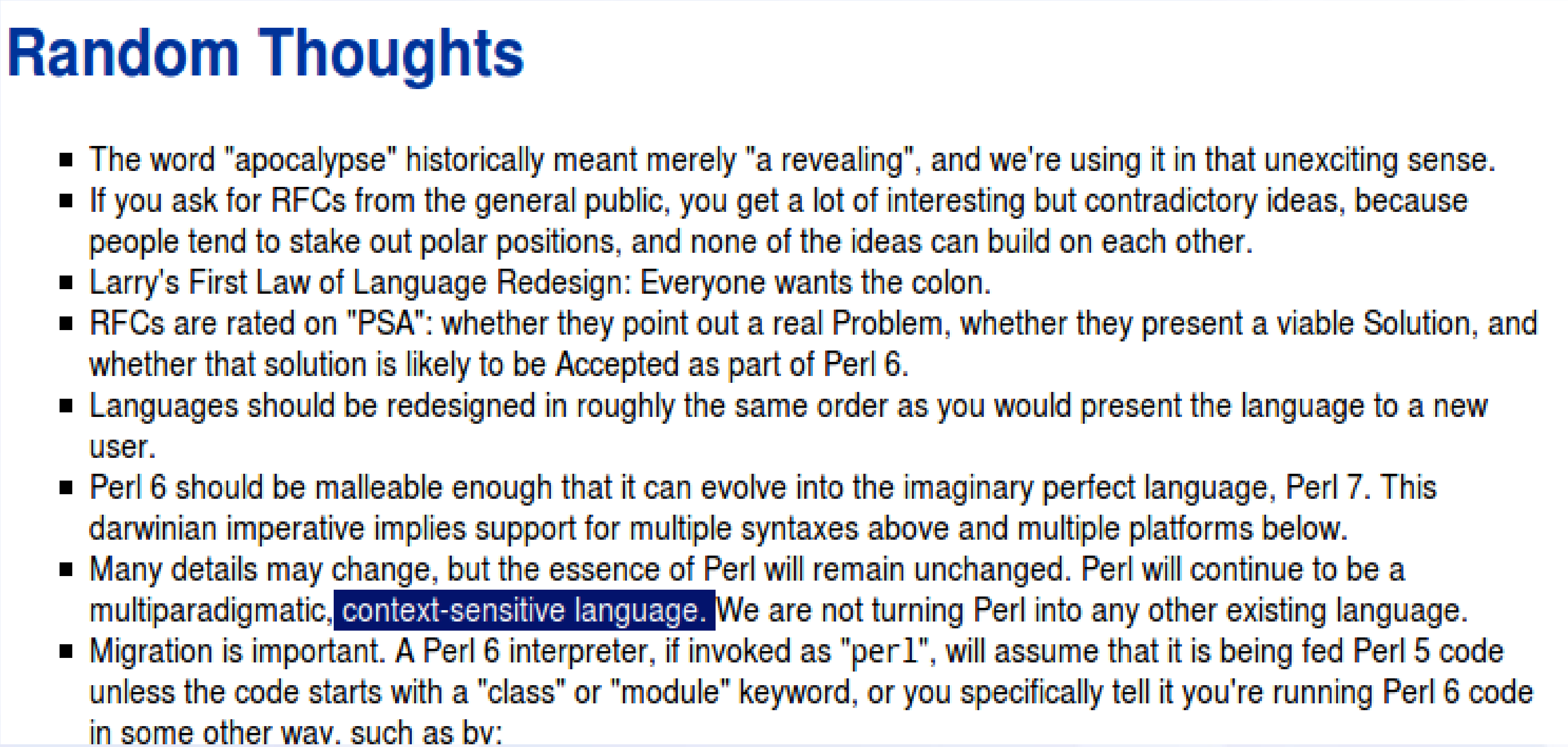

STEPHEN
WOLFRAM
A NEW
KIND OF
SCIENCE
WOLFRAM
A NEW
KIND OF SCIENCE

New P6 Meta Ops:

more direct
right brain
access

End of my Sermon

Synopsis 1: Overview



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:

Perl 5 in context

Perl 5 in context

wantarray

context: **wantarray**

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

Perl 6 in context

no wantarray !!!

P6 Internals

context

=

data type

=

class

Type Classes:

Num

Str

Array

Hash

...

As Known:

```
my $num = 12;
my $str = 'text';
```

Optional:

```
my Num $num = 12;
my Str $str = 'text';
```

How to convert ?

```
my Num $num = 12;
my Str $str = 'text';
```

As Java knows ?

public method to_string {

Not Perl 6:

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

Not Perl 5:

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

Perl 5 in Context

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

Secret Goatse Op

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

No Real List Context

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

Explicit in Perl 6

@() array

Explicit in Perl 5

@{} array

Explicit in Perl 6

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

Perl 6 Major Contex

$	scalar
@	array
%	hash

Invariant Sigil

$	scalar
@	array
%	hash

Invariant Sigil

$scalar

@array

%hash

Don't Show Context

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

Native Hash Slice

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

Sigils

$	scalar
@	positional
%	asociative
&	callable
::	namespace

Context operator

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

With Long Version

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

Braces Optional

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

Item Context

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

List Context

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

P5 List Context

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

Hash Context

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

Code Context

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

Namescpace Context

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

::($str)

More Context Op

~	string
+	numeric
?	boolean

Negative Op

~	string
+ -	numeric
? !	boolean

Example without ()

```
~@list
+@list
?@list
```

String Context

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

Num Context

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

Bool Context

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

Bool Context

?

Bool Context

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Grey is Logic

?   !

?^   ?| ?&

^ | &

//   ^^ || &&   ff   fff

??   !!

Bool Context

?

Bool Context

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

Bool Context

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

```
True
```

Bool Context

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

True

Bool::True in String Context

Bool Context

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

False

Bool::False v String Cont.

Is it so ?

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

```
True
```

Bool::True v String Context

Is it not so ?

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

False

Bool::False v String Cont.

High Precedence

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

High Precedence

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

```
2
```

True in Num Context = 1

Low Precedence

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

Low Precedence

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

```
True
```

46 v Bool Kontext = True

Bool Context

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

```
1
```

46 v Bool Context = True

Bool Context

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

```
1
```

If   unless   while   until

That was easy !

?   !

?^ ?| ?&

^ | &

//   ^^ || && ff   fff

??   !!

Still ?

?^ ?| ?&

Wants Bool Context

?^ ?| ?&

Known Logic Ops

?^ ?| ?&

1+2 = ?

?^ ?| ?&

What could that be?

```
say 0 ?|  'tree';
```

Clearly !!!

```
say 0 ?|  'tree';
```

```
True
```

False or True = True

What could that be?

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

Clear as daylight.

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

```
True
```

True xor False = True

You get a sense

? !

?^ ?| ?&

^ | &

// ^^ || && ff fff

?? !!

Hmmmmm ?

^ | &

Hmmmmm ?

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

Now Know More ?

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

```
any(0, 'tree')
```

Junctions !

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

```
any(0, tree)
```

literally: 0 or 'tree'

Short Overview

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

0 & 1 & 3  =  all(0,1,3);

0 ^ 1 ^ 3    =  one(0,1,3);
```

Quiz Time !

2 ~~ 0 | 1 | 3 | 7

Expected Differently

2 ~~ 0 | 1 | 3 | 7

False

Next Question

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

You get:

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

```
any(False, True, False)
```

Nicer if statements !

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

Junctions !

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

True

Junctions !

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

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

It Gets Clearer

?   !

?^ ?| ?&

^ | &

//   ^^ || && ff   fff

??   !!

No Forced Context

// ^^ || && ff fff

short circuit **OR**

```
doit()   ||   doelse();
```

short circuit **OR**

```
doit()   ||   doelse();
```

```
doit() unless  doelse();
```

Defined OR

```
doit()  //  doelse();
```

Defined OR

```
doit()   //   doelse();
```

```
doelse() unless defined doit();
```

short circuit **AND**

```
doit()  &&  doelse();
```

```
doelse()   if   doit();
```

short circuit **XOR**

```
doit()   ^^   doelse();
```

eXclusive OR

```
doit()   ^^   doelse();
```

```
my($l, $r)=(doit(), doelse());
if not $l { $r }
else      { $r ?? Nil !! $l }
```

No **else** with **unless**

```
    doit()   ^^   doelse();

my($l, $r)=(doit(), doelse());
if not $l { $r }
else      { $r ?? Nil !! $l }
```

All Shortcuts

```
this()   ||   that();
this()   //   that();
this()   &&   that();
this()   ^^   that();
```

Boundary Values

$a min $b

$a max $b

$a minmax $b

Boundary Values

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

Flipflop

```
begin()   ff   end();
begin()   fff   end();
```

Was .. | .... in $ contxt

```
while ... {
    run() if  begin() ff end();
    run() if  begin() fff end();
}
```

Skoro u Cile

?   !

?^ ?| ?&

^ | &

//   ^^ || && ff   fff

??   !!

Ternärer Op

??   !!

Ternary Op

was ? :

?? !!

Ternary Op

was ? :

eval in Bool context

??   !!

Ternary Op

was ? :

eval in Bool context

values unchanged

?? !!

All Clear Now ?

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

Numeric Kontext

+ - * / % %% **

+^ +| +&

+< +>

Everybody knows:

+ - * / % %% **

+^ +| +&

+< +>

Division

7 / 3

7/3(2.333) | 2

Modulo

7 % 3

Modulo

$$7 \% 3$$

$$7 = 3 * 2 + \underline{1}$$

ModMod?

7 %% 3

Indivisible

7 %% 3

False => remainder 1

Numeric Context

+ - * / % %% **

+^ +| +&

+< +>

Bit Logic

+^ +| +&

Bit Logic

(was:)

```
  ^    |    &
 +^   +|   +&
```

Bit - Shift

+< +>

Bit - Shift

(was:)

<< >>

+< +>

Numeric Context

```
+  -  *  /  %  %%  **
+^  +|  +&
+<  +>
```

Someth. Forgotten?

Someth. Forgotten?

++

--

Ordered Sets

++	after
- -	before
	cmp

Ordered Sets

cmp:

Less, Same, More

Ordered Sets

cmp:

Less, Same, More

-1, 0, 1

Still in Context

<=>

leg

cmp

Compare in Context

<=>	Num Context
leg	Str Context
cmp	elsewhere

Compare in Context

<	Num Context
lt	Str Context
before	elsewhere

Compare in Context

>	Num Context
gt	Str Context
after	elsewhere

Ordered Sets

++   1 after

- - 1 before

Equality in Context

==	Num Context
eq	Str Context
===	Id. (typ & val)

Equality in Context

==	Num Context
eq	Str Context
eqv	everywhere

Equality in Context

==	Num Context
=:=	binding
eqv	dynamic

Dynamic in Context

```
if 2 eqv 2.0 {
```

Data Type => Content

```
if 2 eqv 2.0 {
```

Int() vs. Rat()

Data Type => Content

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

Data Type => Content

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

True (Num Kontext)

Numeric Context

```
+  -  *  /  %  %%  **
+^  +|  +&
+<  +>
```

String Context

~

~^ ~| ~&

~~ x

Perlish to_string

~

Was Once .

~

```
say 'combine' ~ 'Watson';
```

String Context

~

~^ ~| ~&

~~ x

Letter Logic

~^ ~| ~&

Letter Logic

1 +| 2 = 3

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

String Context

~

~^ ~| ~&

~~ x

String Context

~

~^ ~| ~&

~~ x

Anyone Knows

```
say '-' x 20;
```

Multiply Strings

```
say '-' x 20;
```

```
'--------------------'
```

String Context

```
      ~
~^  ~|  ~&
     ~~ x
```

List Context

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

Comma Operator

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

Same As:

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

Comma Operator

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

```
Int
```

Comma Operator

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

Comma Operator

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

Comma Operator

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

```
List of Parameter
```

Capture Context

| named parameter

|| positional parameter

List Context

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

Sequence Operator

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

Yadda Operator

```
sub planned { ... }
```

Yadda Operator

```
sub planned { ... }
sub planned { ??? }
sub planned { !!! }
```

Sequence Operator

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

Sequence op can!

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

Range Op can't!

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

Range Op can't!

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

Sequence op can.

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

Sequence Operator

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

Sequence Operator

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

Forgot something?

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

Forgot something ?

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

No List ?

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

```
0..9
```

Depends On Context

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

braces -> precedence

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

```
0..9
```

What is it ?

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

What is it ?

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

Range ???

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

```
Range
```

Range ???

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

Obj with 2 values

Range ???

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

```
True
```

How you create Lists

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

```
List
```

List - Output?

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

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

for - ces List context

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

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

real perlheads do:

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

real perl5heads do:

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

Perl 6 heads:

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

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

List Context

, ... xx X Z

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

Play with Lists

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>;
```

Cartesian Product

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

```
eins dan eins rabauke
zwo dan zwo rabauke
```

Its pairs in real:

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

Its pairs in real:

```
say elems(<1 2>X<3 4>);
```

```
4
```

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 as a Op

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

read write var

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

List Context

, 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_fun($_)] }
  <== @input;
```

Other Direction

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

Append Mode

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

Pointy Sub

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

List Context

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

Meta Ops

= !

X Z R S

[] [\]

<< >>

Meta Op =

```
@sum  += 3;
```

Meta Op !

```
if $age  !< 18 {
```

Meta Op !

```
if $age  !< 18 {

# real P6 code
```

Meta Op R

```
$age  = 2 R- 18;

# == 16
```

Meta Op S

```
$age  = 2 S- 18;

# == -16
```

Meta Op S

```
$age  = 2 S- 18;

# actually error
```

Meta Op S

```
$age  = 2 S- 18;

# don't parallel !!!
```

Meta Op S

```
$age  = 2 S- 18;

# later important
```

Meta Ops

= !

X Z R S

[] [\]

<< >>

Meta Op X

Let's Remember

```
say <1 2> X <a b>
```

```
1 a 1 b 2 a 2 b
```

Let's Remember

<1 2> X <a b>

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

Cartesian Product

<1 2> X <a b>

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

Cartesian Pairs

<1 2> X~ <a b>

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

no num out of 'a'

<1 2> X+ <a b>

Stacktrace

Cartesian Pairs

<1 2> X* <3 4>

Cartesian Pairs

<1 2> X* <3 4>

3, 4, 6, 8

Meta Op Z

# guess what ?

Metaop Z

<1 2> Z~ <3 4>

Metaop Z

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

'13','24'
```

Metaop Z

<1 2> Z* <3 4>

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>
```

Meta Ops

= !

X Z R S

[] [\]

<< >>

Meta Op []

Do it like Gauss

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

Forces List Context

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

[+]  1 .. 100
```

Forces List Context

True

[<] 1 .. 100

Any Clue?

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

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

What's that ?

1, 3, 6

[\+] 1 .. 3

What's that ?

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

[\+] 1 .. 3

Hyper Op <<

Birthday !!!

```
@age >>++;
```

Birthday !!!

```
@age  >>+=>> 1;
```

all get older

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

@age = @age >>+>> 1;

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

only one gets older

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

@age = @age <<+<< 1;

@age == 19;
```

interesting cases

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

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

interesting cases

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

ERROR

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

19, 24, 36

complexity ++

~~

not today

Thank You !!!