Herbert Breunung

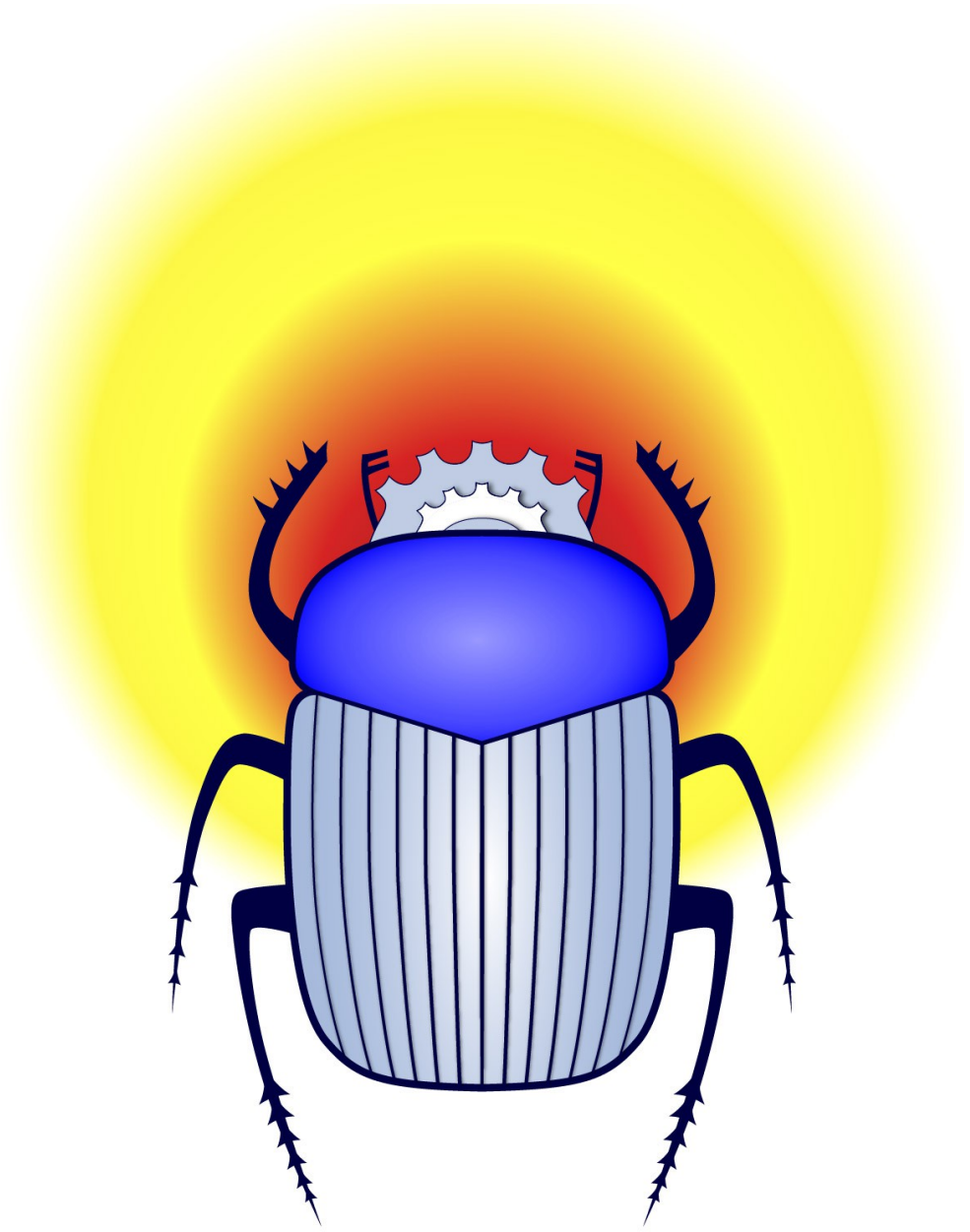
writing a P6 module

Herbert Breunung

write moderate

speak program

no comment



P6::Math::Matrix

P6::Math::Matrix

Aus dem Leben eines Perl 6 Modulautors

content :

- syntax
- conventions
- naming

Perl 5

P6: grand design

P6 not baroque

english style

visual axis

like home

use name;

require name;

almost like home

use name;

need name;

require name;

modules

use v6.c;

unit module name;

class name::space{...}

what is unit?

unit class Math::Matrix..;
unit role ...;
unit module ...;

version und source

```
unit class Math::Matrix
:ver<0.1.8>
:auth<github:pierre-vigier>;
```

own module

```
use AttrX::Lazy;
```

own type

```
subset PosInt of Int where * > 0;
```

does less

```
subset PosInt of Int where * > 0;

$rows where * > 0,
```

best

```
subset PosInt of Int where * > 0;

$rows where * > 0,

PosInt $rows,
```

better as

@rows **where** .all ~~ Numeric,

type like a pro

```
subset NumList of List
        where .all ~~ Numeric;

fail "..."
    unless @rows ~~ NumList;
```

2D

@rows **where** .all ~~ Numeric,

all(@m[*;*]) ~~ Numeric;

band aid solution

```
submethod check_index
(Int $row, Int $col) {,
```

kinds of methods

```
method name          # normal
method !name         # private
submethod name       # greedy
sub name             # no OO
```

default constructor

```
method new (@m) {
    check_matrix_data( @m );
     self.bless( rows => @m);
}

Math::Matrix.new([[1]]);
```

why not ?

```
Math::Matrix.new: q:to/END/;
1 4 6 4 1
1 8 2 6 1
1 6 2 8 1
END
```

why not ?

```
multi method new (@m) {
    check_matrix_data( @m );
     self.bless( rows => @m);
}
```

another constructor ...

```
multi method new (Str $m) {
    check_matrix_data( @m );
     self.bless( rows => @m);
}
```

another constructor !

```
multi method new (Str $m) {
    my @m = $m.lines.map:
   {[ .words.map: *.Numeric ]};
    check_matrix_data( @m );
    self.bless( rows => @m);
}
```

finds the right type

```
multi method new (Str $m) {
    my @m = $m.lines.map:
        { [ .words.map:
            { .Numeric } ] };
    check_matrix_data( @m );
    self.bless( rows => @m);
}
```

why not ?

```
Math::Matrix.new: q:to/END/;
        1 4 6 4 1
        1 8 2 6 1
        1 6 2 8 1
    END
```

nothing special

```
method new (@m) {
    check_matrix_data( @m );
     self.bless( rows => @m);
}

Math::Matrix.new([[1]]);
```

is special

```
submethod BUILD (...) {
    @!rows = ;
}
```

neutral element

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

neutral element

```
method new-identity
  (Math::Matrix:U: Int $I -->
    self.bless(...);
}
```

type objekt

```
method new-identity
 (Math::Matrix:U: Int $I -->
   self.bless(...);
}
```

type objekt

```
method new-identity
 (Math::Matrix:U: Int $I -->
   self.bless(...);
}
Math::Matrix.new-identity(3);
```

type objekt

```
method new-identity
 (Math::Matrix:D: Int $I -->
   self.bless(...);
}
$matrix.new-identity(3);
```

zero

0 0 0

0 0 0

0 0 0

zero

```
method new-zero
 (Math::Matrix:U:
  PInt $rows, PInt $rows -->
   self.bless(...);
}
```

diagonal

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{matrix}$$

constructor

default new(....)

fast, simple, no prop. calc.

special new-....()

diff. signat. some propert.

custom BUILD

called by both

norm

default p q

p, q optional

special names

tested in signature

MMD

every norm - own multi

p-q Norm

```
multi method norm
( PInt :$p = 2, PInt :$q = 1
--> Numeric) { ...
```

Norm

```
multi method norm
(Str $w where * eq 'row-sum'
                    --> Numeric) {
    max map {
        [+] map *.abs, @$_
    }, @!rows;
}
```

cloning

```
method clone {
  self.bless(rows =>@!rows)
}
```

no deep clones

```
method clone {
  self.bless(rows =>@!rows)
}

# b/c data ro – but  we could
```

marshal

```
multi method perl
 (Math::Matrix:D: --> Str) {
    self.WHAT.perl  ~
  ".new(" ~ @!rows.perl ~ ")";
}
```

multi is important

```
multi method perl
 (Math::Matrix:D: --> Str) {
    self.WHAT.perl  ~
  ".new(" ~ @!rows.perl ~ ")";
}
```

? $matrix

```
method Bool
 (Math::Matrix:D: --> Bool) {
    ! self.is-zero;
}
```

is ~~ boolean context

```
method is-square
    (Math::Matrix:D: --> Bool) {
 $!column-count == $!row-count
}
```

+ $matrix

```
method Numeric
 (Math::Matrix:D: --> Bool) {
    self.elems
}
```

~ $matrix

```
method Str
    (Math::Matrix:D: --> Str) {
    join (' | ', @!rows.map: .Str})
}
```

```
1 2 | 3 4
```

say $matrix

```
method gist
  (Math::Matrix:D: --> Str) {
    ...
}
```

```
1 2
3 4
```

say Math::Matrix

```
multi method gist
(Math::Matrix:U: --> Str)
{ "({self.^name})" }

(Math::Matrix)    # like (Int)
```

called gist 4 reason

method gist (...) { ...}

1 2 3 4 5 ..
6 7 8 9 10 ..

...

say $matrix.full

method full (...) { … }

1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16
17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41

implicit: $a == $b

```
method equal
(Math::Matrix:D:
 Math::Matrix $b --> Bool) {
    @!rows ~~ $b!rows
}
```

free bonus: $a eqv $b

```
method equal
(Math::Matrix:D:
 Math::Matrix $b --> Bool) {
    @!rows ~~ $b!rows
}
```

smartmatch: $a ~~ $b

```
method ACCEPTS
(Math::Matrix:D:
 Math::Matrix $b --> Bool) {
    self.equal( $b );
}
```

$a ~~ Math::Matrix

```
multi method ACCEPTS
(Math::Matrix:D:
 Math::Matrix $b --> Bool) {
    self.equal( $b );
}
```

$a.cell-type ~~ Int

```
multi method !built_..
(Math::Matrix:D: -->Numeric) {
    return Complex
        if any(@!rows[*;*])
            ~~ Complex; ... ...
}
```

$matrix.map({...})

```
method map
 (Math::Matrix:D: &coderef
             --> Math::Matrix:D:) {
    Math::Matrix.new (
       [ @!rows.map:
          {[ $_.map( &coderef ) ]}])
}
```

Matrix → Matrix

```
$matrix.map(...
$matrix.list.map(...
$matrix.list-rows.flat.map(...
$matrix.list-rows.map(...
$matrix.list-columns.map(...
```

List → List

Matrix → Matrix

```
$matrix.splice-rows(...
$matrix.splice-columns(...

$matrix.grep(...
```

sets cells you don't greped to 0

$m.reduce-rows(&[+])

method reduce ...

method reduce-rows ...

method reduce-columns ...

treat Matrix like a list

```
$matrix.list     # elems as list
$matrix.elems    # nr of elems

$matrix.cont     # element of
                 # even ranges
```

enable List contex

```
$matrix.list     # elems as list
$matrix.elems  # nr of elems

$matrix.cont   # element of
                    # even ranges
```

set ops need lists

1 contained in matrix:

$matrix (cont) 1

1 is element of matrix:

1 (elem) $matrix

set ops need lists

1 contained in matrix:

$matrix (cont) 1

$matrix.cont(1)

$matrix.cont(1..2.5)

bonus 4 list method

all value are in range:

(1..3) (>=) $matrix

some value are in range:

(1..3) (<=) $matrix

UTF ops bonus

all value are in range:

(1..3) (я) $matrix

some value are in range:

(1..3) (ю) $matrix

operators

sub name **is export** { … }

Op: + - *

```
multi sub infix:<+>
  (Math::Matrix:D $a,
   Math::Matrix:D $b -->
   Math::Matrix:D) is export {
    $a.add($b);
}
```

$m2 + $matrix

```
multi sub infix:<+>
  (Math::Matrix:D $a,
   Math::Matrix:D $b -->
   Math::Matrix:D) is export {
    $a.add($b);
}
```

other ops

- $matrix

$matrix ** 2

op alias

$a dot $a

$a ק $a

$a ÷ $a

tensor product

$a x $a

$a ⌐ $a

L2 norm: || $matrix ||

```
multi sub circumfix:<|| ||>
  (Math::Matrix $a --> Numeric)
   is equiv(&prefix:<!>)
   is export  {
    $a.norm()
}
```

MM [[1, 2], [3, 4]]

```
multi sub prefix:<MM>
    (Array $m --> ::?CLASS:D)
is tighter(&postcircumfix:<[]>)
            is export (:MM) {
    ::?CLASS.new(@$m)
}
```

pitfalls

safety lock

```
use lib '.';
use Math::Matrix;
```

simple port

eval "...";

allright …

EVAL "…";

safety lock

```
use MONKEY;
use MONKEY-SEE-NO-EVAL;

EVALFILE $file;
EVAL "...";
```

name spaces

```
class Math::Matrix.. {}
role … {}
module … {}
```

forgot unit

unit class Math::Matrix..;
unit role ...;
unit module ...;

Semicolon !

```
sub g { 9.81 }
sub e { 2.718 }  # incl.

sub g { 9 }; sub d { 2 }
```

routines

g() + d() : 11

g() - d() : 7

without & and ()

g() + d : 11

g() - d : 7

ambiguous !

g + d : Err

g - d : Err

unambiguous !

```
g + d()    : 11
g - d()    :  7
```

space !

g+d : 11

g-d : Err

secial: -

$!row-count - 1
$!row-count-1
$!row-count-d

conversionen

```
method column ...
@!rows.keys.map:{
    @!rows[$_;$column]
};
```

Seq vs List

```
method column
(Math::Matrix:D:
 Int:D $column --> List) {
    @!rows.keys.map:{
        @!rows[$_;$column]
    };
```

Seq vs List

```
method column
(Math::Matrix:D:
 Int:D $column --> List) {
    (@!rows.keys.map:{
        @!rows[$_;$column]
    }).list;
```

THANK YOU

FIN

lichtkind.de/vortrag