Pharo:
A Little Journey in the Smalltalk Syntax

S. Ducasse

http://www.pharo.org

vmware

esug
EUROPEAN SMALLTALK USER GROUP

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Yesplan
Let's make it happen

2denker

MAD
ENVIRONNEMENT

BetaNine
software engineering

Sensus HighOctane

Pleiad

Mines de Douai
LILLE EUROREGION

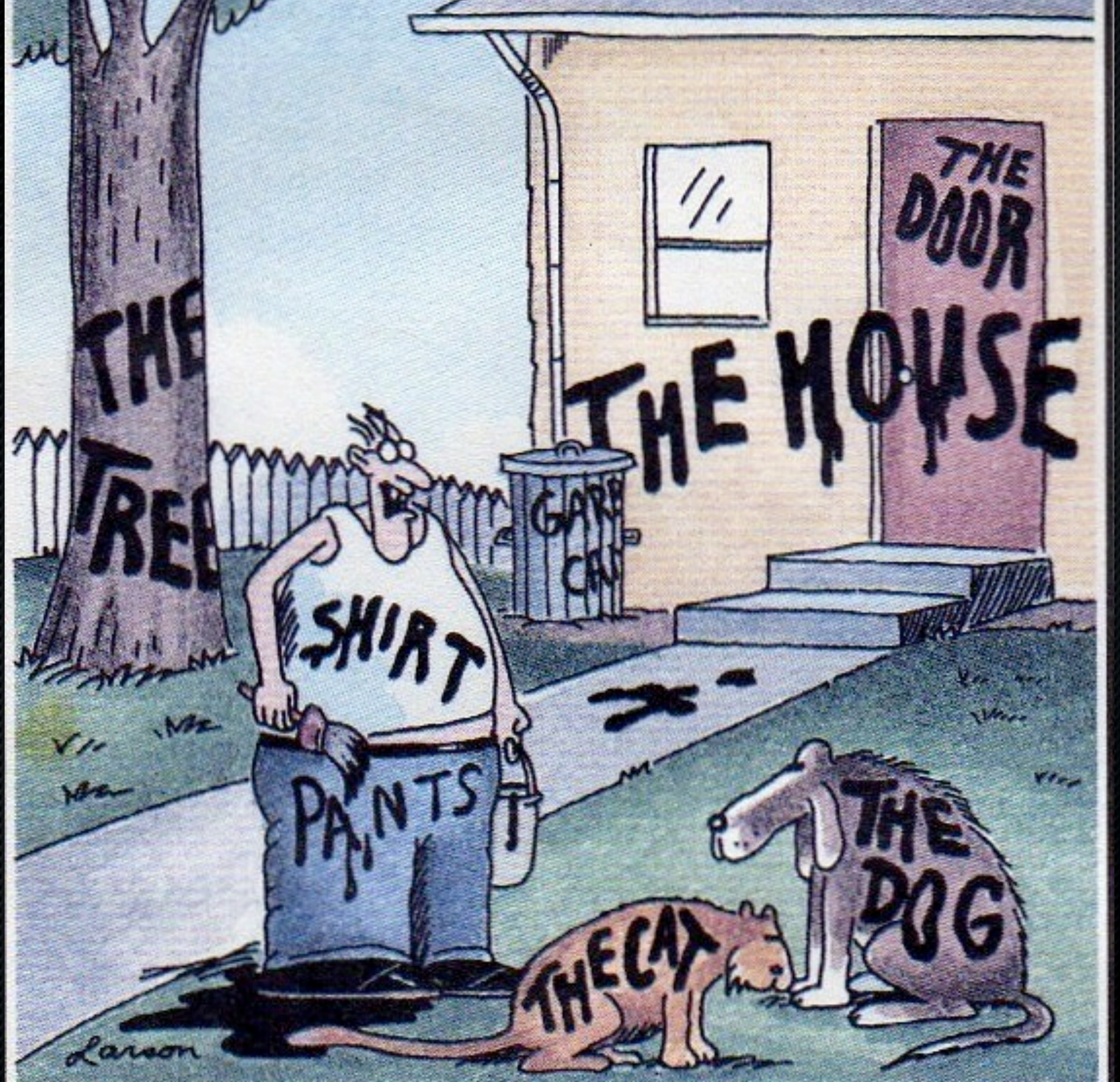
UNIVERSITÄT BERN

RMod

Show you that this is simple

a piece of cake

a first appetizer


THE TREE
THE HOUSE
THE DOOR
GARBAGE CAN
SHIRT
PANTS
THE CAT
THE DOG
Larson

Yeah!

Smalltalk is a dynamically typed language

```
ArrayList<String> strings
                = new ArrayList<String>();
```

strings := ArrayList new.

```
Thread regThread = new Thread(
        new Runnable() {
          public void run() {
                this.doSomething();} });
      regThread.start();
```

[self doSomething] fork.

Smalltalk = Objects +
Messages + (...)

Roadmap

Fun with numbers

1 class

1 class

>SmallInteger

1 class maxVal

```
1 class maxVal
>1073741823
```

1 class maxVal + 1

1 class maxVal + 1

>1073741824

(1 class maxVal + 1) class

(1 class maxVal + 1) class

>LargePositiveInteger

(1/3) + (2/3)

$(1/3) + (2/3)$

>1

2/3 + 1

$2/3 + 1$

$> 5/3$

1000 factorial

1000 factorial

402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969
404800479988610197196058631666872994808558901323829669944590997424504087073759918823627727188732
519779505950995276120874975462497043601418278094646496291056393887437886487337119181045825783647
849977012476632889835955735432513185323958463075557409114262417474349347553428646576611667797396
668820291207379143853719588249808126867838374559731746136085379534524221586593201928090878297308
431392844403281231558611036976801357304216168747609675871348312025478589320767169132448426236131
412508780208000261683151027341827977704784635868170164365024153691398281264810213092761244896359
928705114964975419909342221566832572080821333186116811553615836546984046708975602900950537616475
847728421889679646244945160765353408198901385442487984959953319101723355556602139450399736280750
137837615307127761926849034352625200015888535147331611702103968175921510907788019393178114194545
257223865541461062892187960223838971476088506276862967146674697562911234082439208160153780889893
964518263243671616762179168909779911903754031274622289988005195444414282012187361745992642956581
746628302955570299024324153181617210465832036786906117260158783520751516284225540265170483304226
143974286933061690897968482590125458327168226458066526769958652682272807075781391858178889652208
164348344825993266043367660176999612831860788386150279465955131156552036093988180612138558600301
435694527224206344631797460594682573103790084024432438465657245014402821885252470935190620929023
136493273497565513958720559654228749774011413346962715422845862377387538230483865688976461927383
814900140767310446640259899490222221765904339901886018566526485061799702356193897017860040811889
729918311021171229845901641921068884387121855646124960798722908519296819372388642614839657382291
123125024186649353143970137428531926649875337218940694281434118520158014123344828015051399694290
153483077644569099073152433278288269864602789864321139083506217095002597389863554277196742822248
757586765752344220207573630569498825087968928162753848863396909959826280956121450994871701244516
461260379029309120889086942028510640182154399457156805941872748998094254742173582401063677404595
741785160829230135358081840096996372524230560855903700624271243416909004153690105933983835777939
410970027753472000000000000000000000000000000000000000000000000000000000000000000000000000000000

1000 factorial / 999 factorial

1000 factorial / 999 factorial

> 1000

10 @ 100

10 @ 100

(10 @ 100) x

10 @ 100

(10 @ 100) x

> 10

```
10 @ 100

(10 @ 100) x
> 10

(10 @ 100) y
```

10 @ 100

(10 @ 100) x

> 10

(10 @ 100) y

>100

Points!

Points are created using @

Puzzle

(10@100) + (20@100)

Puzzle

(10@100) + (20@100)

>30@200

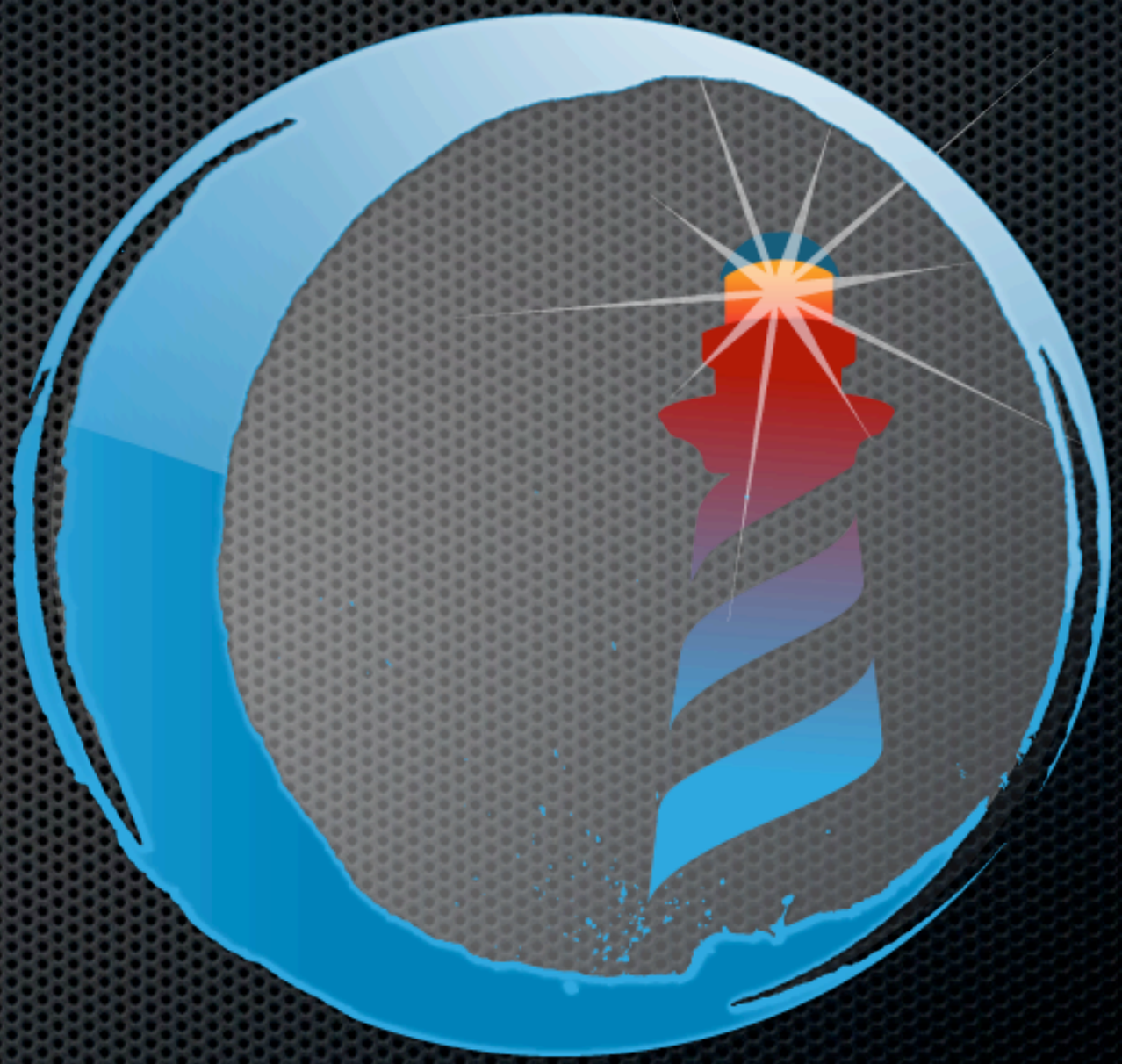
Puzzle

(10@100) + (20@100)

>30@200

Roadmap

Fun with characters,
strings, arrays



$C $h $a $r $a $c $t $e $r

$F, $Q $U $E $N $T $i $N

Characters? :)

space tab cr ... ?!

Character space

Character tab

Character cr

‘Strings’

‘Tiramisu’

Characters

```
12 printString

> '12'
```

Strings are collections of chars

‘Tiramisu’ at: 1

Strings are collections of chars

```
'Tiramisu' at: 1

> $T
```

A program! -- finding the last char

A program!

```
| str |
```

A program!

| str |

local variable

A program!

| str |

str := ‘Tiramisu’.

local variable

A program!

| str |

str := ‘Tiramisu’.

local variable

assignment

A program!

```
| str |
str := ‘Tiramisu’.
str at: str length
```

local variable

assignment

A program!

| str |	local variable
str := ‘Tiramisu’.	assignment
str at: str length	message send

> $u

double ‘ to get one

‘L”Idiot’

> one string

For concatenation use ,

‘Calvin’ , ‘ & ‘, ‘Hobbes’

For concatenation use ,

‘Calvin’ , ‘ & ‘, ‘Hobbes’

> ‘Calvin & Hobbes’

For concatenation use ,

‘Calvin’ , ‘ & ‘, ‘Hobbes’

> ‘Calvin & Hobbes’

Symbols: #Pharo

#Something is a symbol

Symbol is a unique string in the system

#Something == #Something

> true

“Comment”

“what a fun language lecture.

I really liked the desserts”

#(Array)

#(‘Calvin’ ‘hates’ ‘Suzie’)

#(Array)

```
#('Calvin' 'hates' 'Suzie') size
```

#(Array)

```
#('Calvin' 'hates' 'Suzie') size
> 3
```

First element starts at 1

#(‘Calvin’ ‘hates’ ‘Suzie’) at: 2

First element starts at 1

#(‘Calvin’ ‘hates’ ‘Suzie’) at: 2

> ‘hates’

at: to access, at:put: to set

#(‘Calvin’ ‘hates’ ‘Suzie’) at: 2 put: ‘loves’

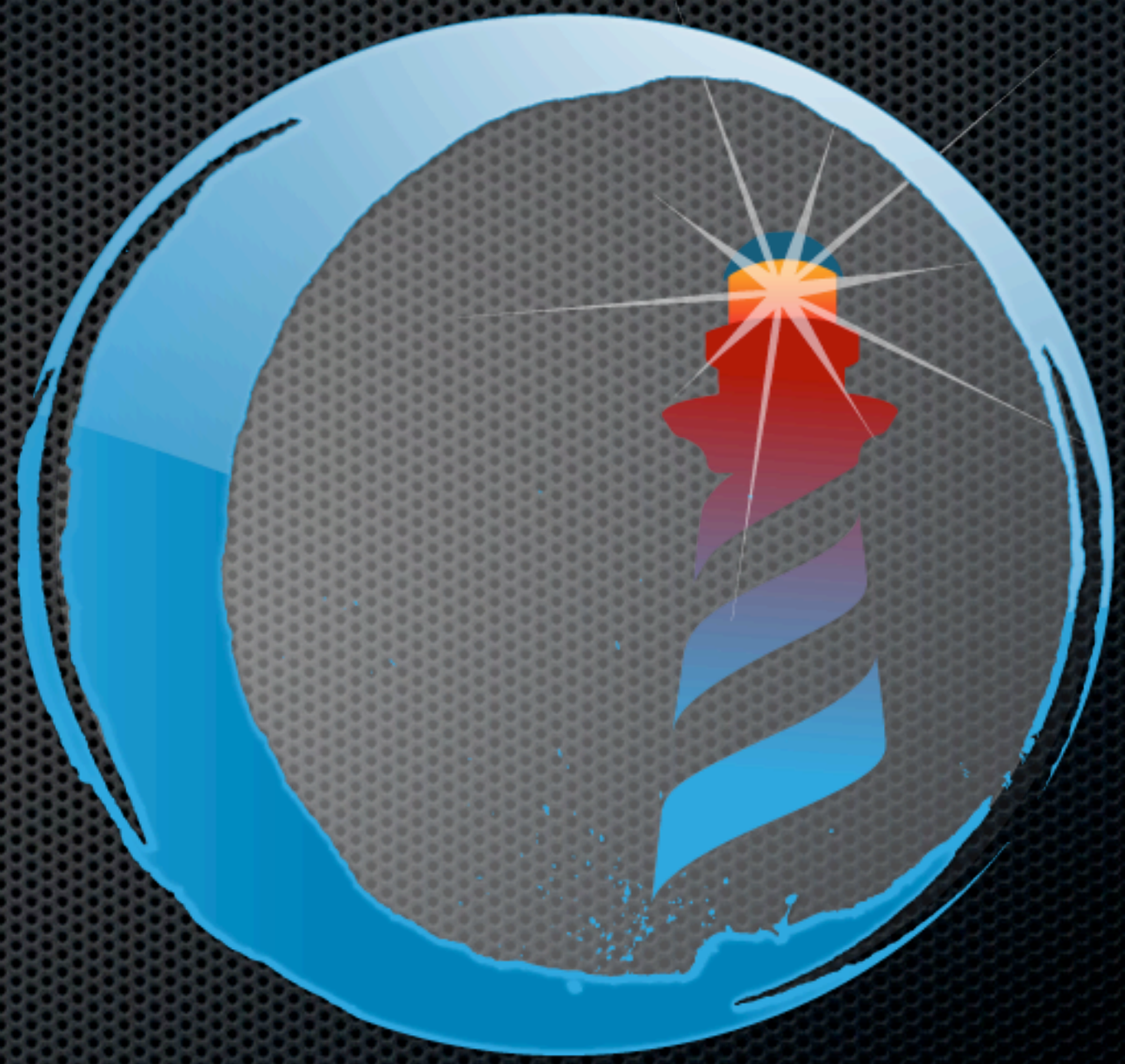
#(Array)

#(‘Calvin’ ‘hates’ ‘Suzie’) at: 2 put: ‘loves’

> #(‘Calvin’ ‘loves’ ‘Suzie’)

Roadmap

Fun with class definitions



A class definition!

Superclass subclass: #**Class**

instanceVariableNames: **'*a b c*'**

...

category: 'Package name'

A class definition!

Object* subclass: #*Point

instanceVariableNames: '*x y*'

classVariableNames: ''

poolDictionaries: ''

category: '*Graphics-Primitives*'

A class definition!

Object* subclass: #*Point

instanceVariableNames: '*x y*'

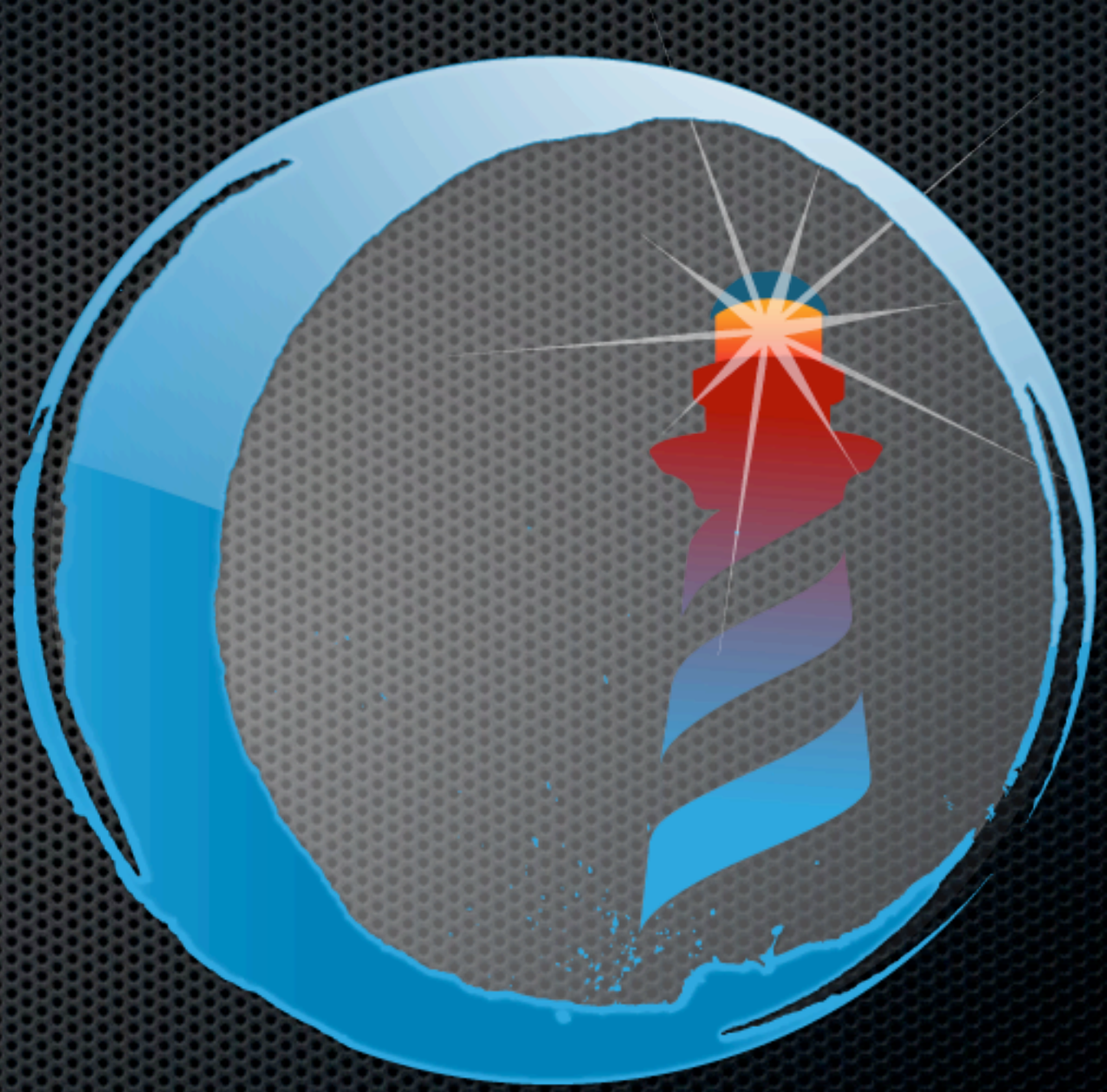
classVariableNames: ''

poolDictionaries: ''

category: '*Graphics-Primitives*'

Roadmap

Fun with methods



On Integer

asComplex

"Answer a Complex number that represents value of the the receiver."

^ Complex real: self imaginary: 0

On Boolean

xor: ***aBoolean***

"Exclusive OR. Answer true if the receiver is not equivalent to aBoolean."

^(self == ***aBoolean***) not

Summary

self, super

can access instance variables

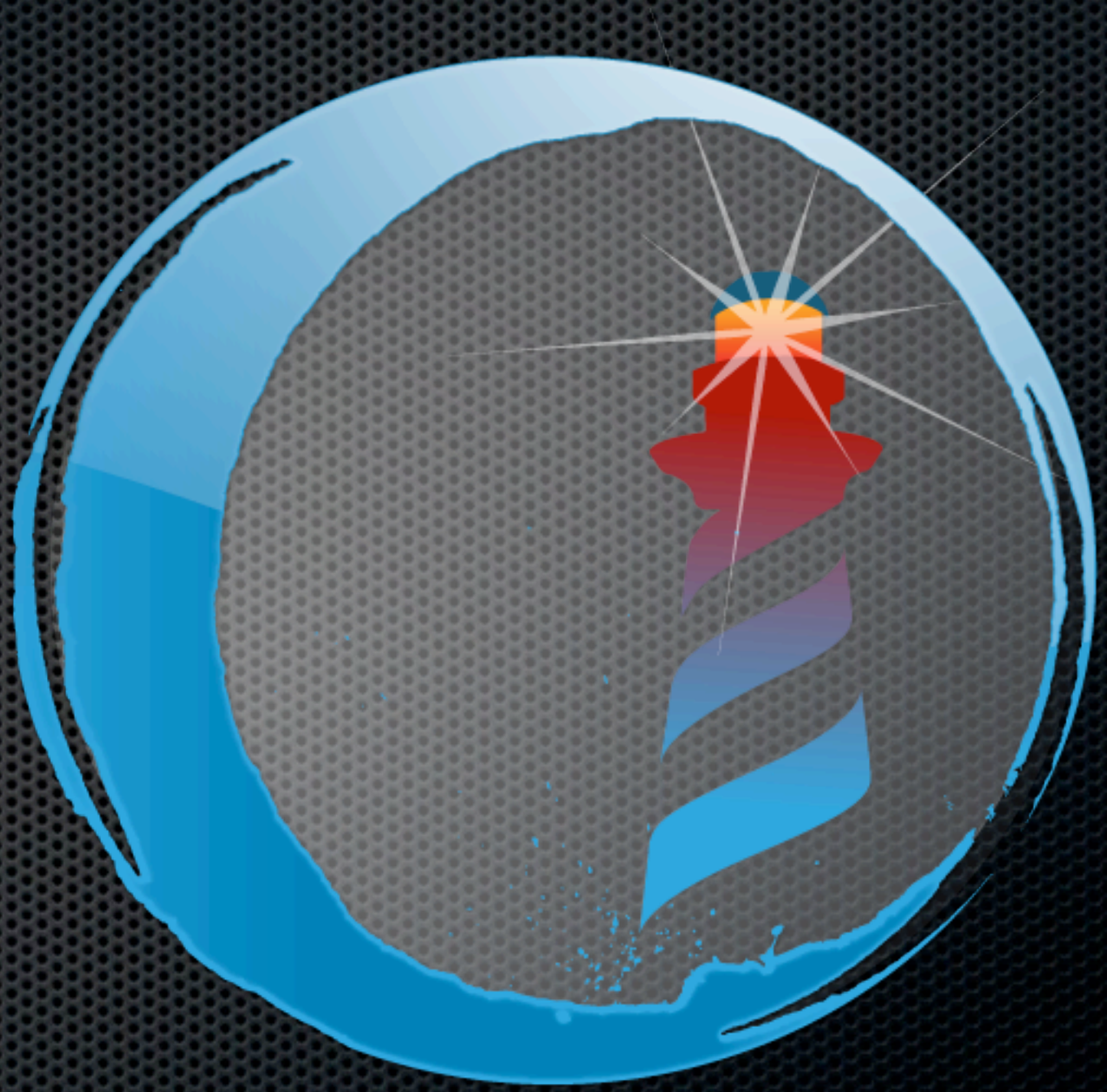
can define local variable | ... |

Do not need to define argument types

^ to return

Roadmap

Fun with unary messages



1 class

1 class

> SmallInteger

false not

```
false not
> true
```

Date today

Date today

> 24 May 2009

Time now

Time now

> 6:50:13 pm

Float pi

```
Float pi

> 3.141592653589793
```

We sent messages to objects or classes!

1 class

Date today

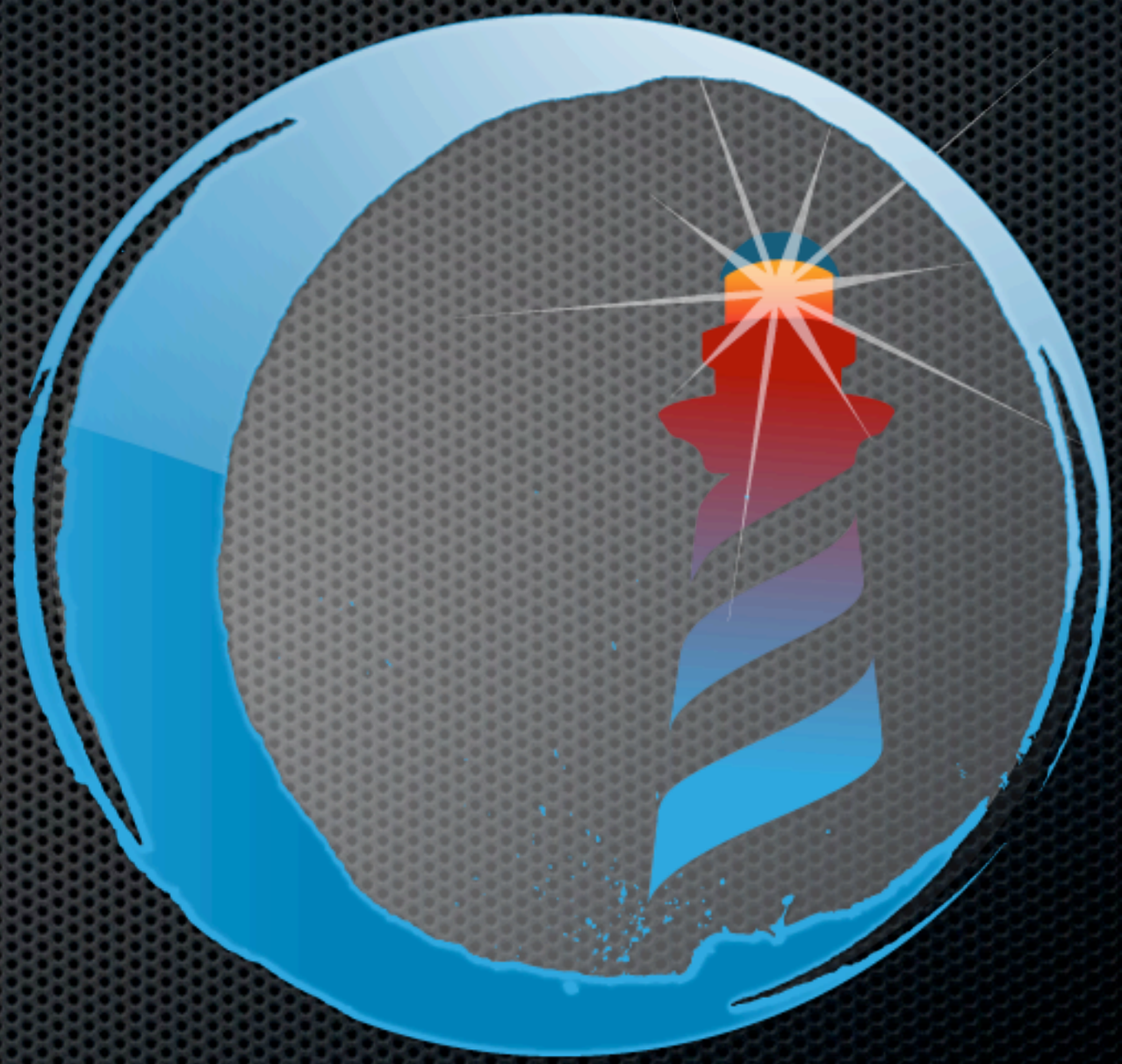
We sent messages to objects or classes!

1 class

Date today

Roadmap

Fun with binary messages



aReceiver aSelector anArgument

Used for arithmetic, comparison and logical operations

One or two characters taken from:

+ - / \ * ~ < > = @ % | & ! ? ,

1 + 2

1 + 2

>3

2 => 3

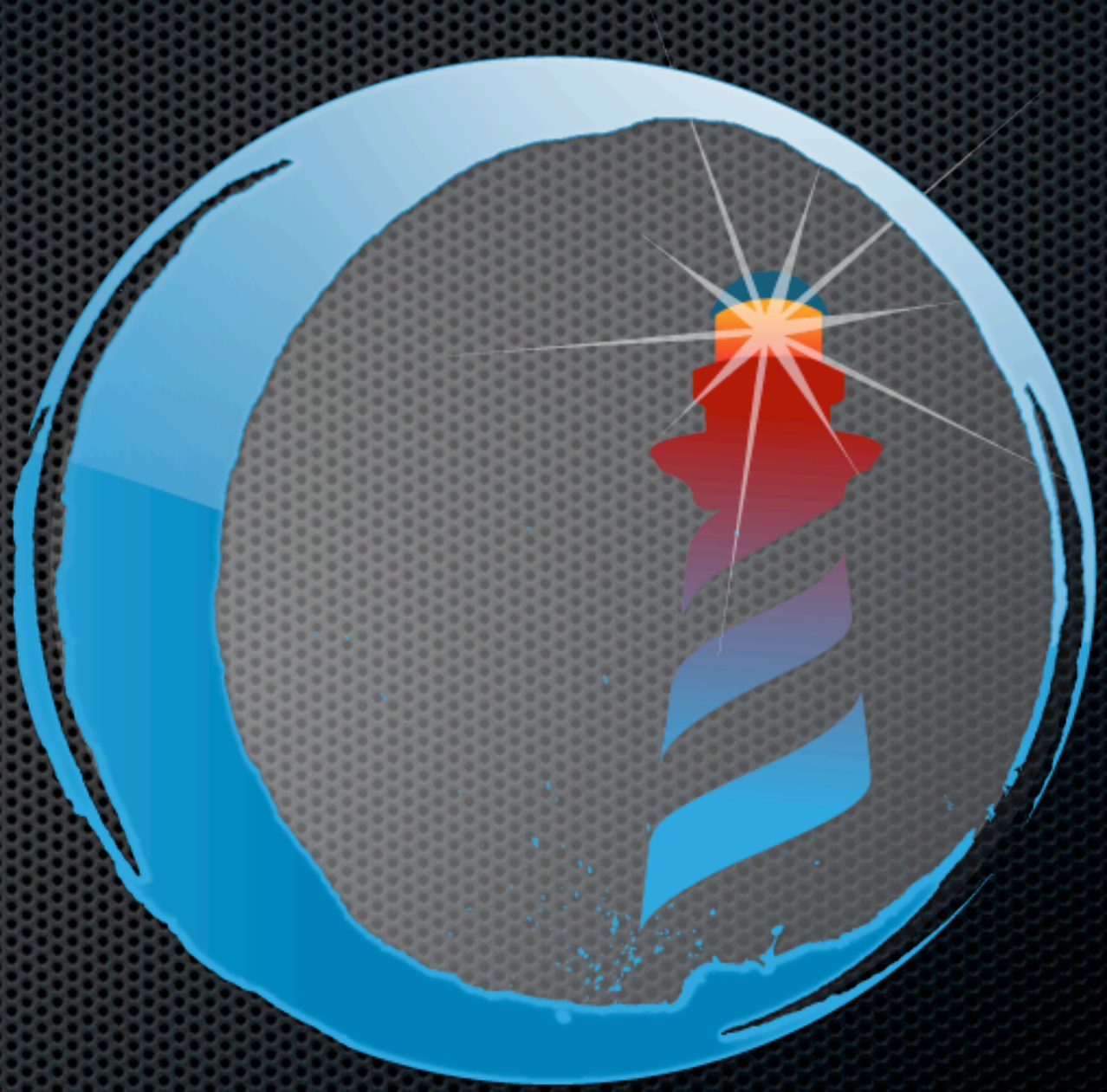
```
2 => 3

> false
```

10 @ 200

‘Black chocolate’ , ‘ is good’

Roadmap

Fun with keyword-based messages



Keyword-based messages

```
ar := #('Calvin' 'hates' 'Suzie').

arr at: 2 put: 'loves'
```

Keyword-based messages

ar := #(‘Calvin’ ‘hates’ ‘Suzie’).

arr at: 2 put: ‘loves’

somehow like arr.atput(2,‘loves’)

From Java to Smalltalk

```
postman.send(mail,recipient);
```

Removing

```
postman.send(mail,recipient);
```

Removing unnecessary

postman send mail recipient

But without losing information

postman send mail to recipient

```
postman.send(mail,recipient);
```

```
postman send: mail to: recipient
```

postman.send(mail,recipient);

postman send: mail to: recipient

#send:to: is the message selector

10@20 setX: 2

10@20 setX: 2

> 2@20

12 between: 10 and: 20

```
12 between: 10 and: 20

> true
```

receiver

keyword1: argument1

keyword2: argument2

equivalent to

receiver.keyword1keyword2(argument1, argument2)

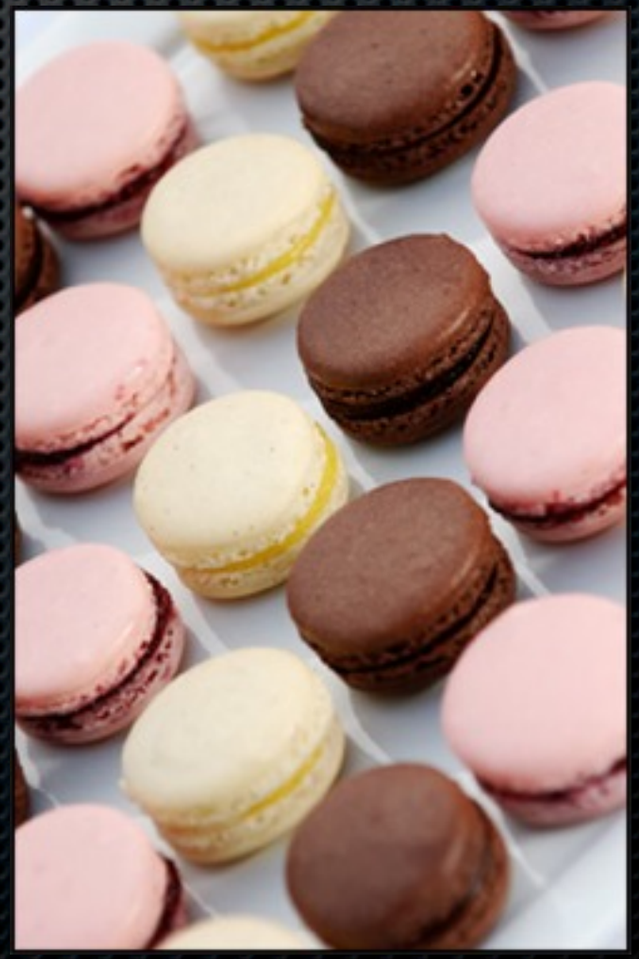
receiver

keyword1: argument1

keyword2: argument2

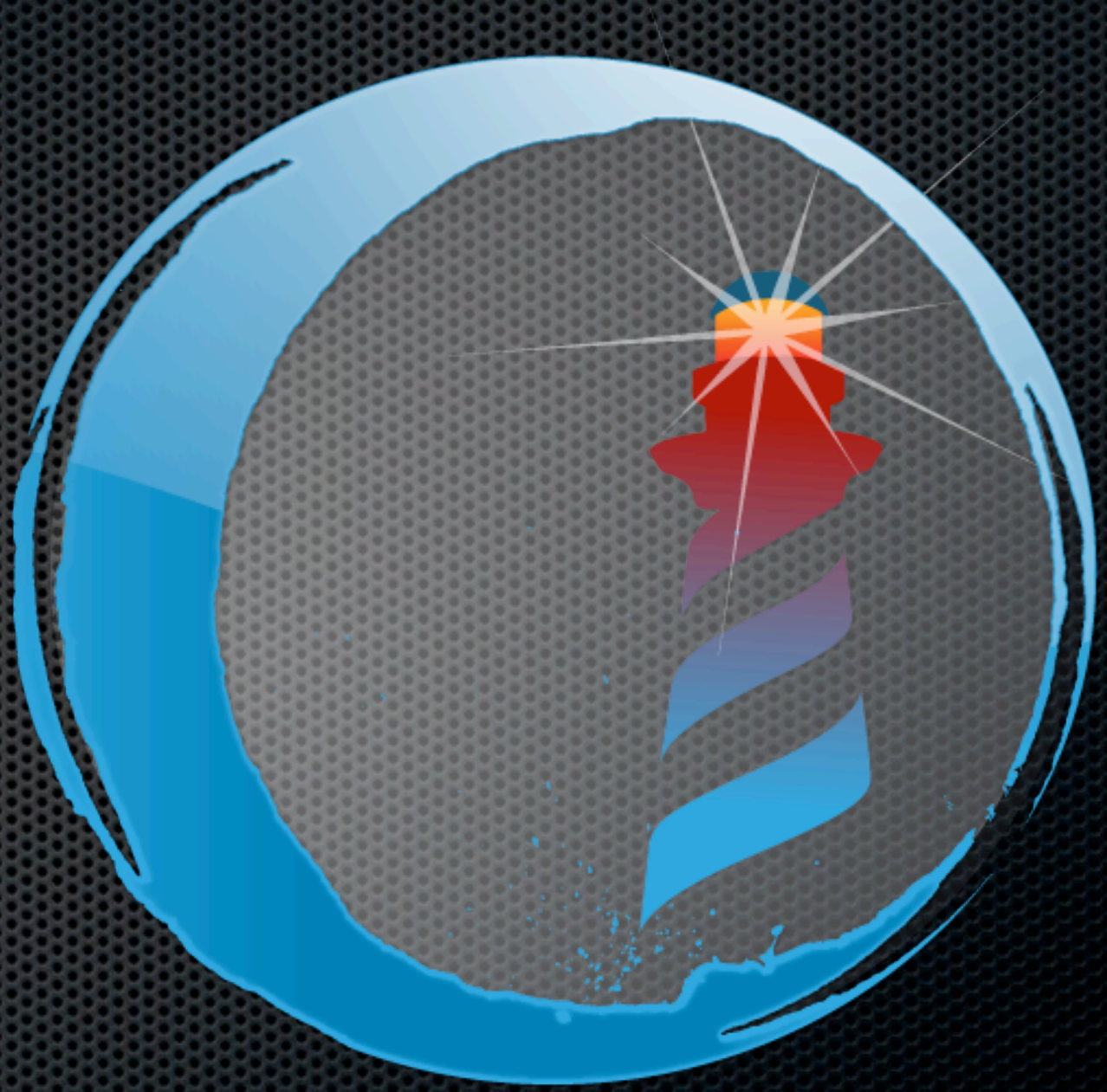
equivalent to

receiver.keyword1keyword2(argument1, argument2)



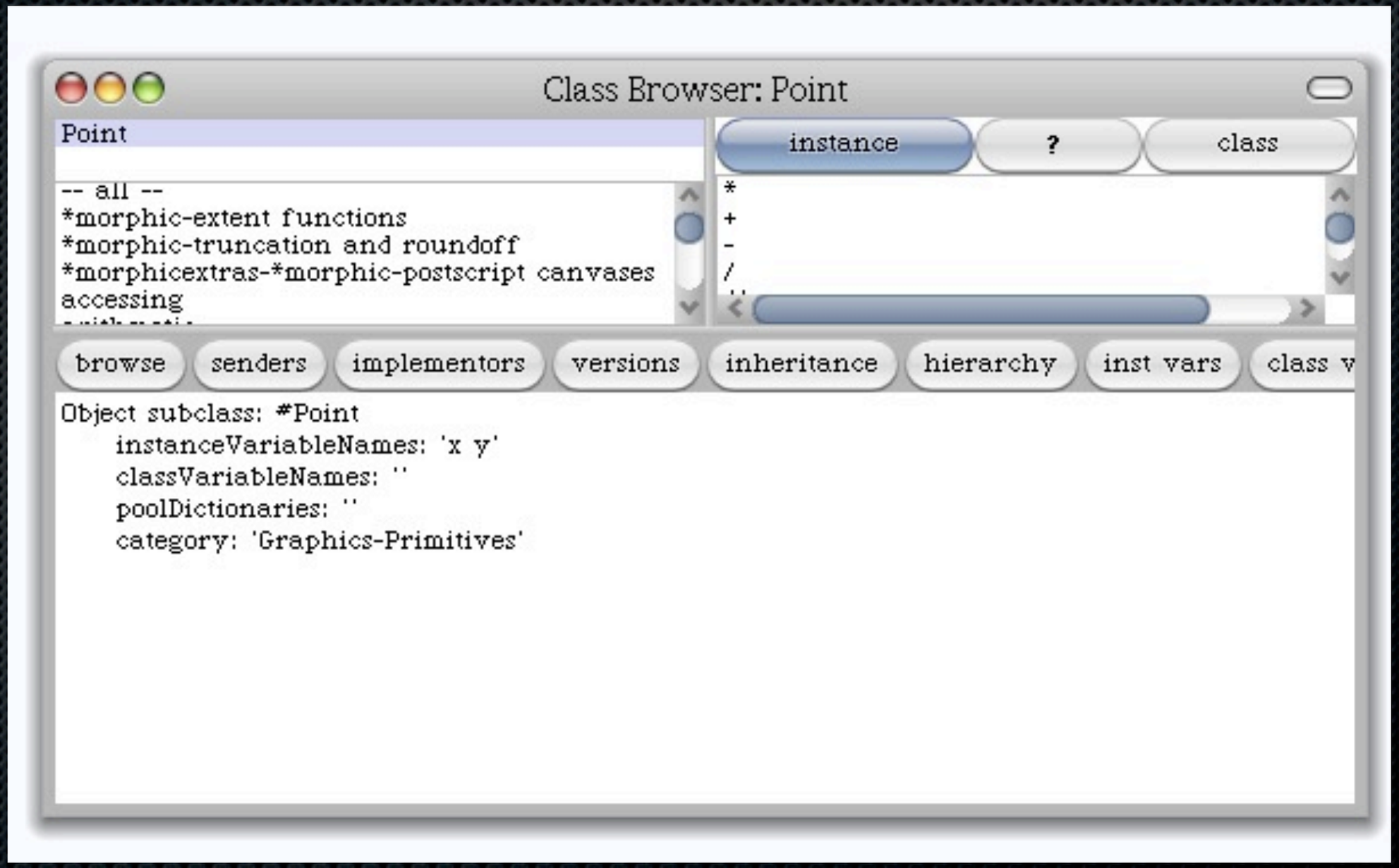
Roadmap

Doing/Printing



Browser newOnClass: Point

Browser newOnClass: Point



Yes there is a difference between

doing (side effect)

and returning an object

```
Browser newOnClass: Point

> a Browser
```

Doing and do not care of the returned result

Browser newOnClass: Point

Doing and really want to see the result!

10@20 setX: 2

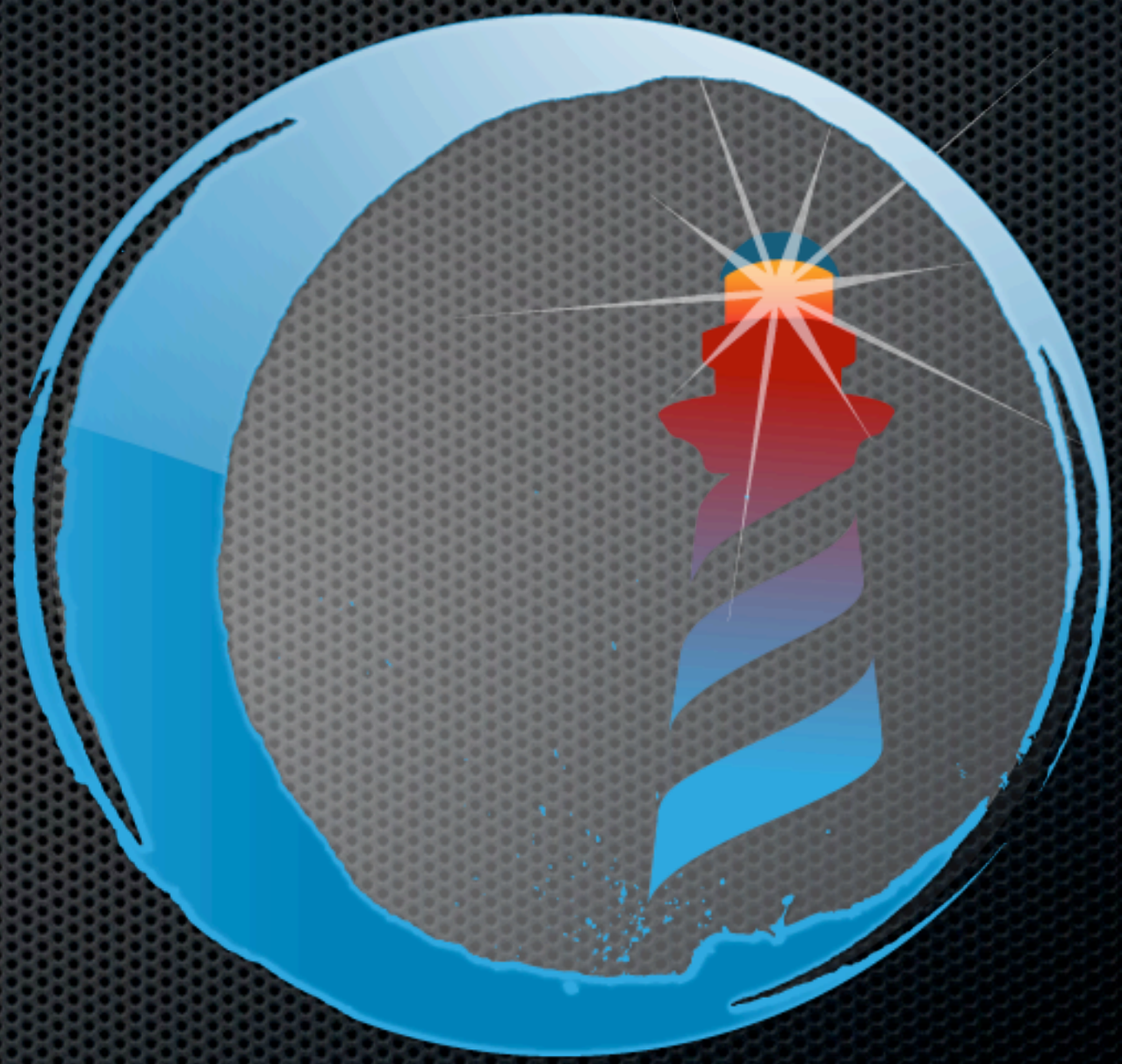
> 2@20

Doing vs printing (doing + print result)

Doing vs printing (doing + print result)

Roadmap

Messages messages
and messages again



Composition: from left to right!

Yes there are only messages

unary

binary

keywords

Composition: from left to right!

69 class inspect

69 class superclass superclass inspect

Precedence

Unary> Binary> Keywords

Precedence

2 + 3 squared

Precedence

```
2 + 3 squared
> 2 + 9
```

Precedence

```
2 + 3 squared
> 2 + 9
> 11
```

Color gray - Color white = Color black

```
Color gray - Color white = Color black

> aColor = Color black
```

```
Color gray - Color white = Color black

> aColor = Color black

> true
```

2 raisedTo: 3 + 2

```
2 raisedTo: 3 + 2

> 2 raisedTo: 5
```

```
2 raisedTo: 3 + 2
> 2 raisedTo: 5
> 32
```

No mathematical precedence

1/3 + 2/3

No mathematical precedence

1/3 + 2/3

>7/3 /3

(Msg) > Unary> Binary> Keywords

Parenthesized takes precedence!

(0@0 extent: 100@100) bottomRight

```
(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint) bottomRight
```

```
(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint) bottomRight
> aRectangle bottomRight
```

```
(0@0 extent: 100@100) bottomRight
> (aPoint extent: anotherPoint) bottomRight
> aRectangle bottomRight
> 100@100
```

0@0 extent: 100@100 bottomRight

```
0@0 extent: 100@100 bottomRight
> Message not understood
> 100 does not understand bottomRight
```

No mathematical precedence

3 + 2 * 10

No mathematical precedence

```
3 + 2 * 10

> 5 * 10
```

No mathematical precedence

3 + 2 * 10

> 5 * 10

> 50

argh!

No mathematical precedence

3 + (2 * 10)

No mathematical precedence

```
3 + (2 * 10)
> 3 + 20
```

No mathematical precedence

3 + (2 * 10)

> 3 + 20

> 23

No mathematical precedence

```
1/3 + 2/3
> 7/3 /3
```

No mathematical precedence

1/3 + 2/3

> (7/3) /3

> 7/9

No mathematical precedence

(1/3) + (2/3)

No mathematical precedence

(1/3) + (2/3)

> 1

Only Messages

(Msg) > Unary > Binary > Keywords

from left to right

No mathematical precedence

Only Messages

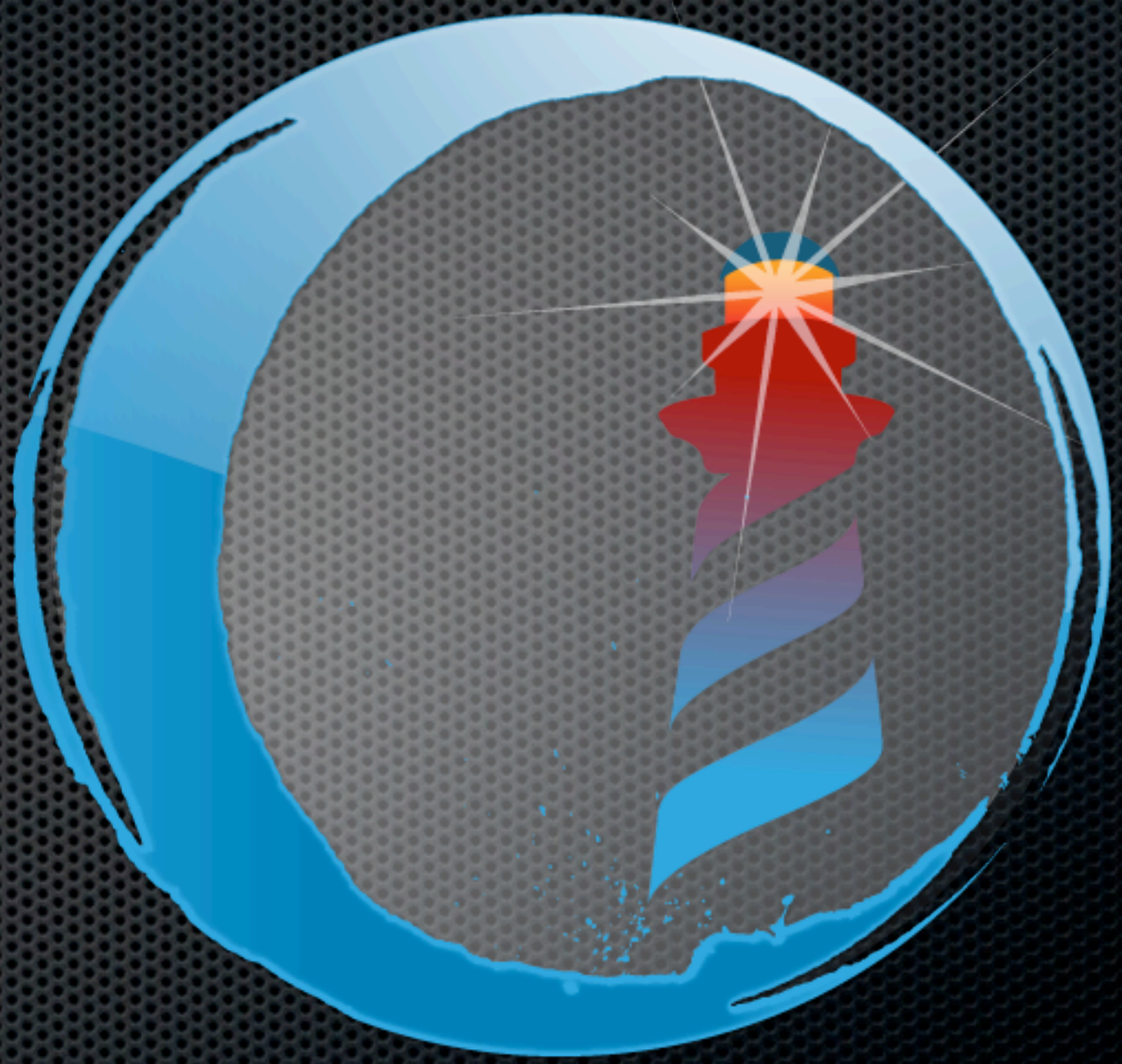
(Msg) > Unary > Binary > Keywords

from left to right

No mathematical precedence

Roadmap

Fun with blocks



Function definition

fct(x) = x * x + x

Function Application

fct (2) = 6

fct (20) = 420

Function definition

```
fct(x) = x * x + x

|fct|

fct:= [:x | x * x + x].
```

Function Application

fct (2) = 6

fct (20) = 420

fct value: 2

> 6

fct value: 20

Other examples

```
[2 + 3 + 4 + 5] value
[:x | x + 3 + 4 + 5 ] value: 2
[:x :y | x + y + 4 + 5] value: 2 value: 3
```

Block

anonymous method

[:*variable1* **:***variable2* **|**
| tmp |
expression1.
...*variable1* ... **]**

value: ...

Block

anonymous method

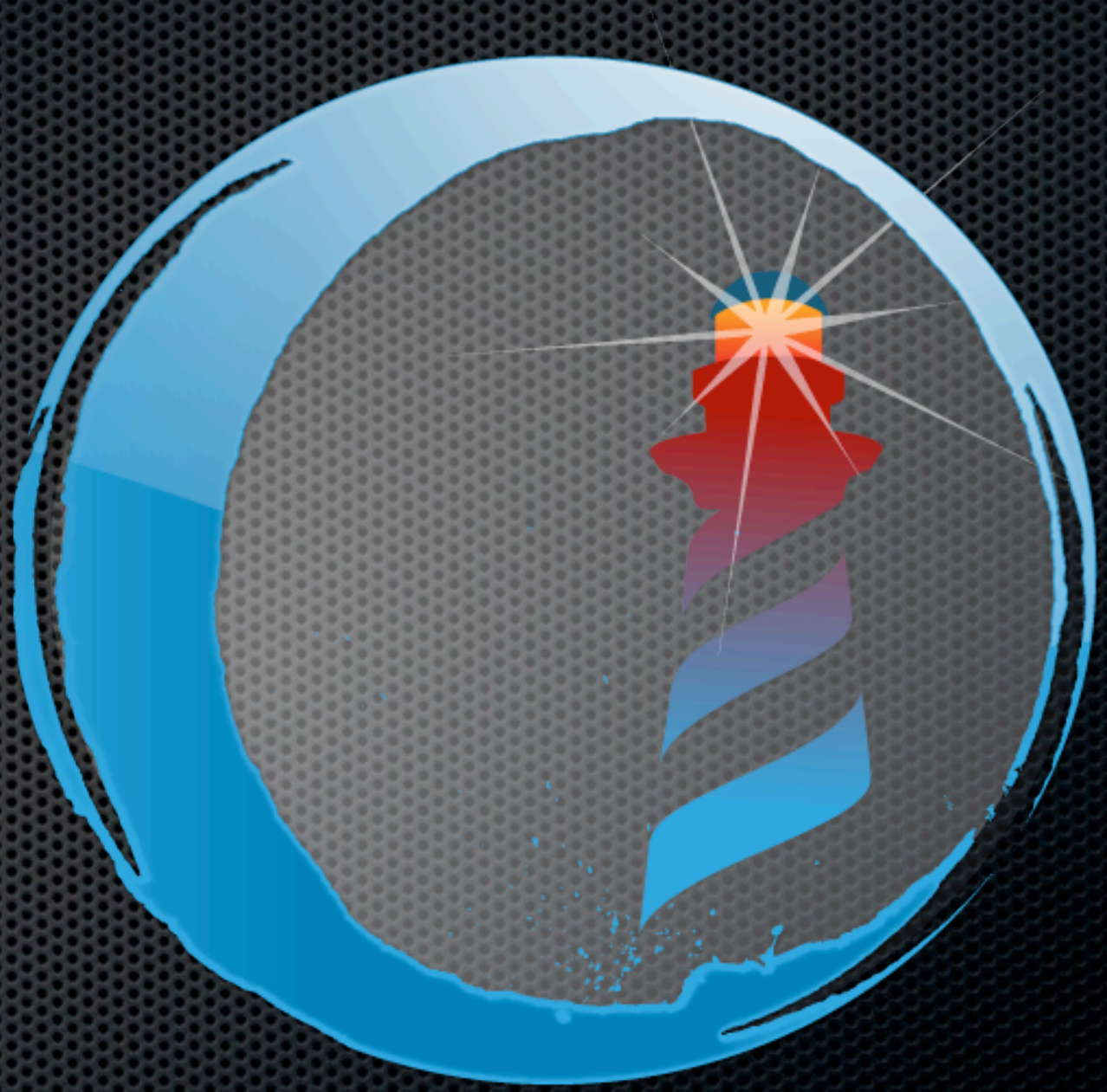
Really really cool!

Can be passed to methods, stored in instance variables

[:*variable1* **:***variable2* **|**
| tmp |
expression1.
...*variable1* ... **]**

Roadmap

Fun with conditional



Example

```
3 > 0
ifTrue:['positive']
ifFalse:['negative']
```

Example

```
3 > 0
    ifTrue:['positive']
    ifFalse:['negative']
```

> ‘positive’

Yes ifTrue:ifFalse: is a message!

```
Weather isRaining
    ifTrue: [self takeMyUmbrella]
    ifFalse: [self takeMySunglasses]
```

ifTrue:ifFalse is sent to an object: a boolean!

Booleans

& | not

or: and: (lazy)

xor:

ifTrue:ifFalse:

ifFalse:ifTrue:

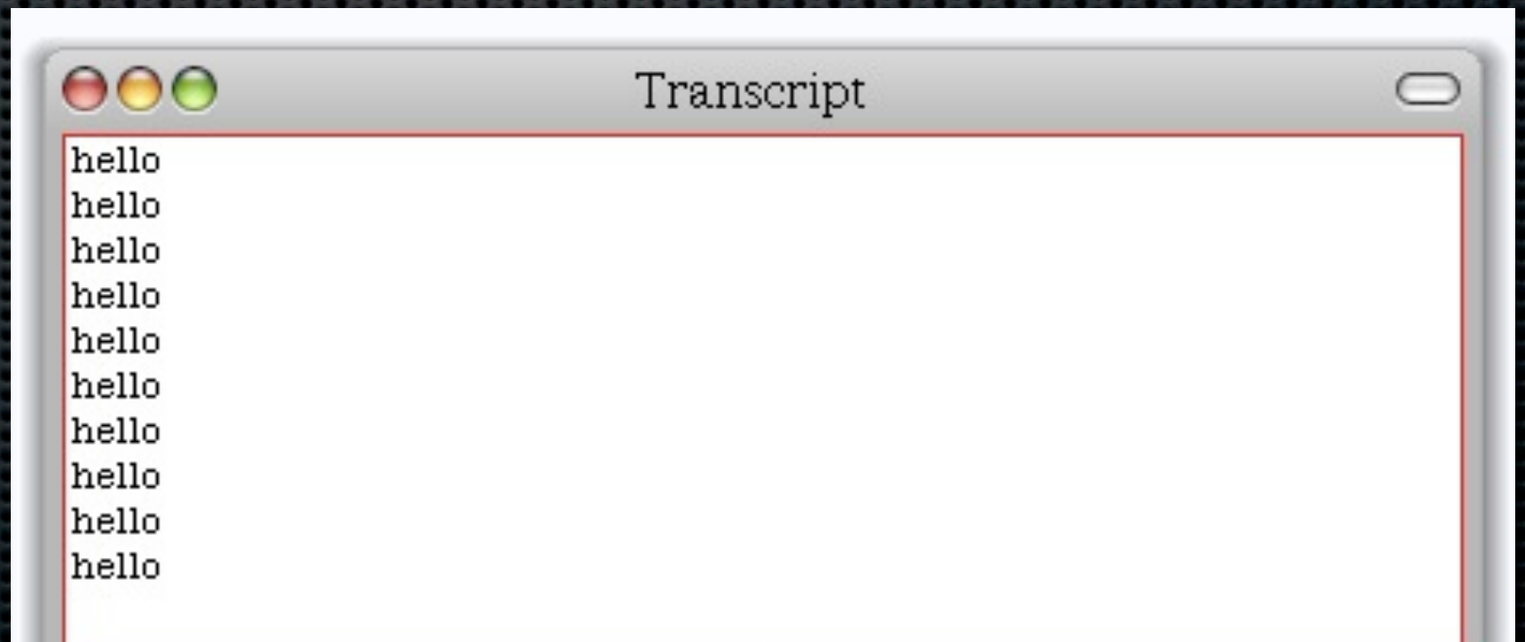
...

Yes! ifTrue:ifFalse: is a message send to a Boolean.

But optimized by the compiler :)

```
10 timesRepeat: [ Transcript show:
'hello'; cr]
```

10 timesRepeat: [Transcript show: 'hello'; cr]



```
[x<y] whileTrue: [x := x + 3]
```

aBlockTest whileTrue

aBlockTest whileFalse

aBlockTest whileTrue: *aBlockBody*

aBlockTest whileFalse: *aBlockBody*

anInteger timesRepeat: *aBlockBody*

Confused with () and [] ?

Only put [ ] when you do not the number of times something may be executed

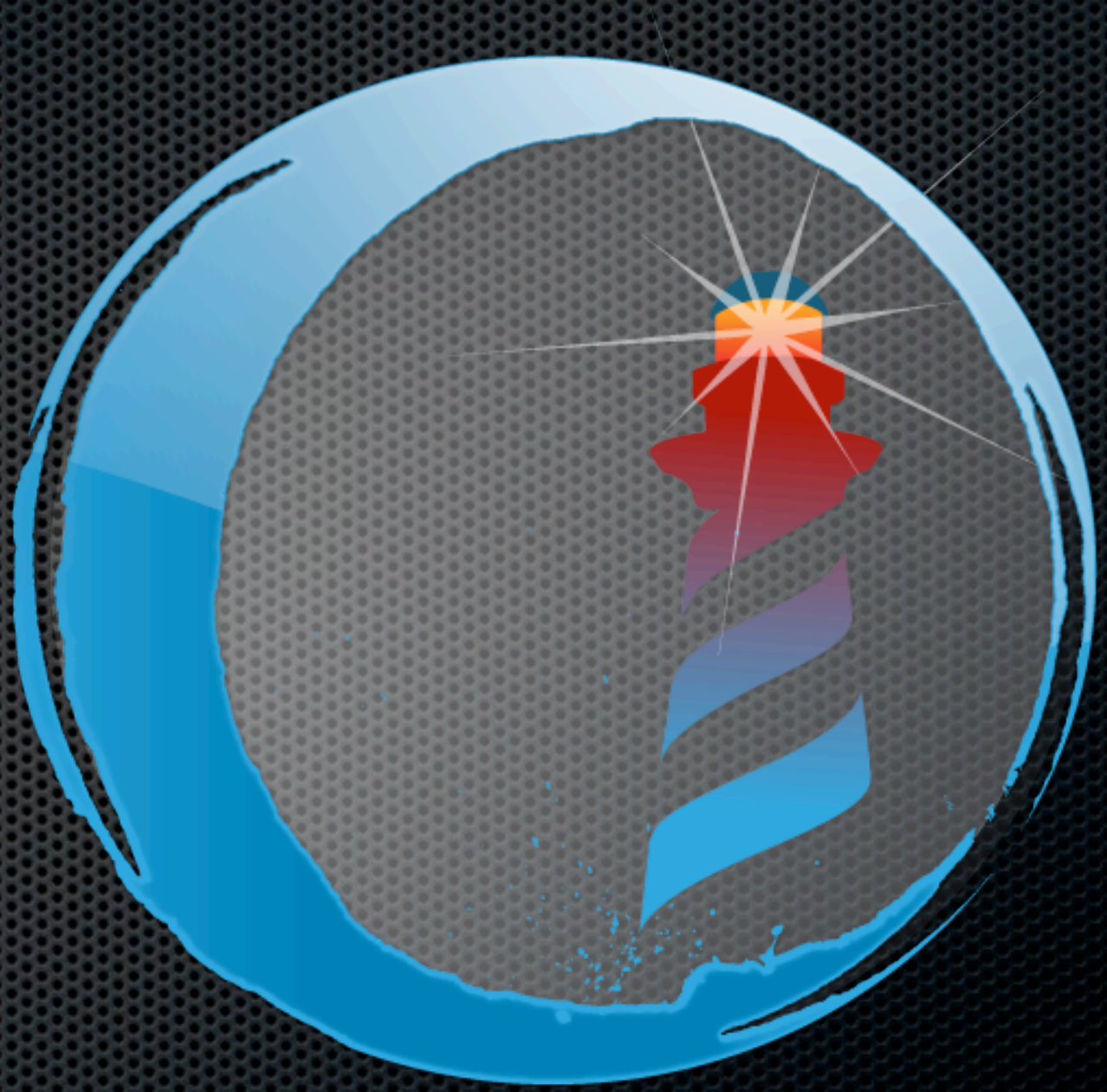
```
(x isBlue) ifTrue: [ x schroumph ]

10 timesRepeat: [ self shout ]
```

Conditions are messages sent to boolean

(x isBlue) ifTrue: [ ]

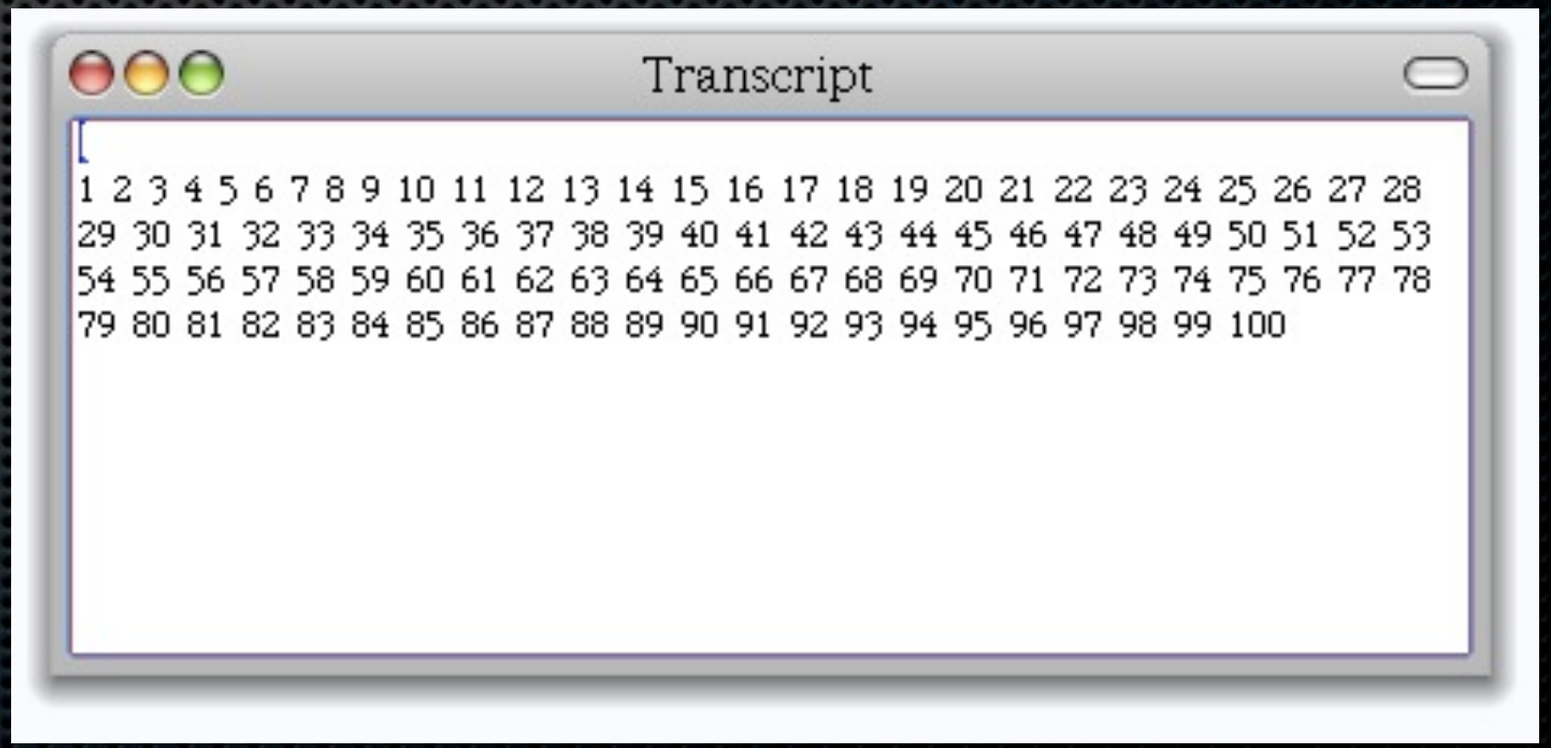
Roadmap

Fun with loops



```
1 to: 100 do:
    [ :i | Transcript show: i ; space]
```

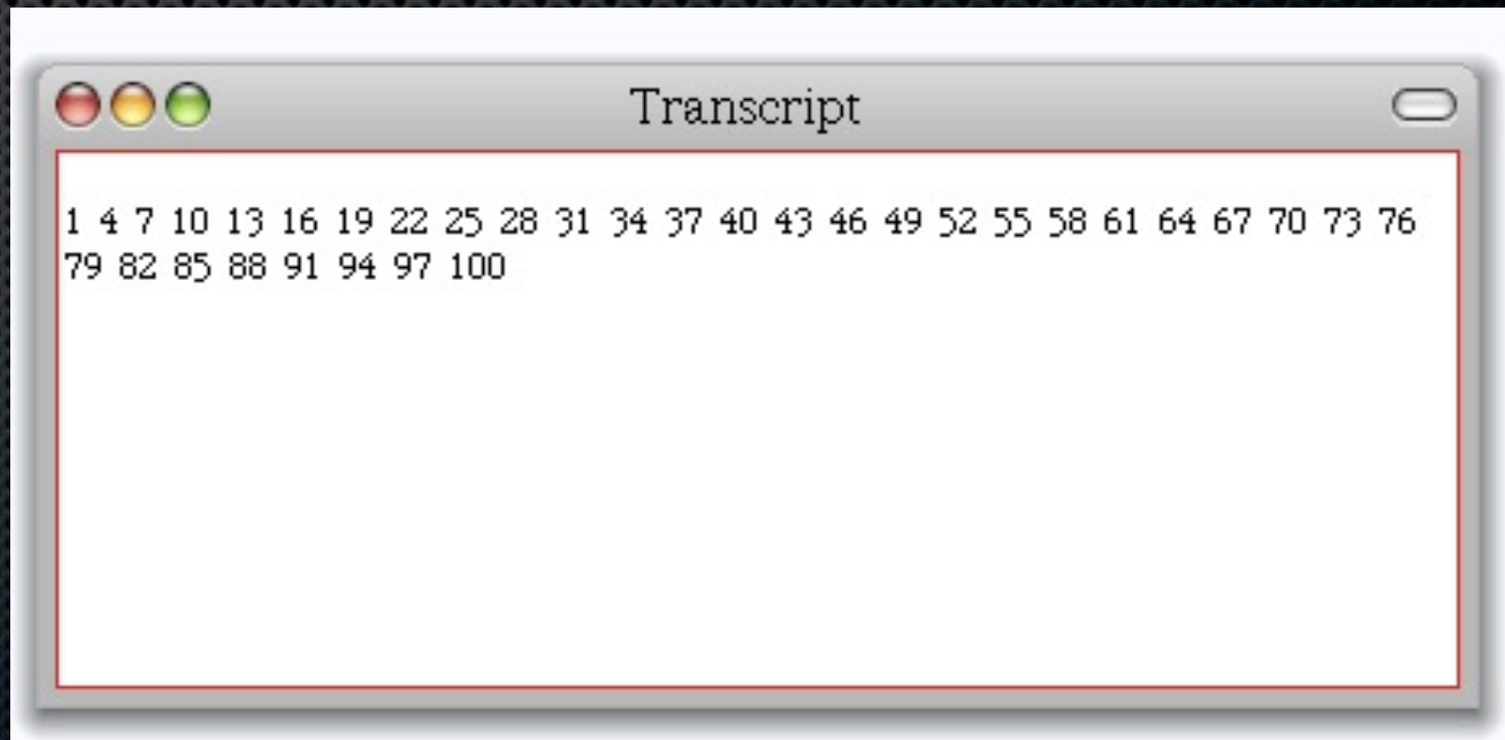
```
1 to: 100 do:
[ :i | Transcript show: i ; space]
```



```
1 to: 100 by: 3 do:
    [ :i | Transcript show: i ; space]
```

1 **to:** 100 **by:** 3 **do:**

[:i | Transcript show: i ; space]



So yes there are real loops in Smalltalk!

to:do:

to:by:do:

are just messages send to integers

So yes there are real loops in Smalltalk!

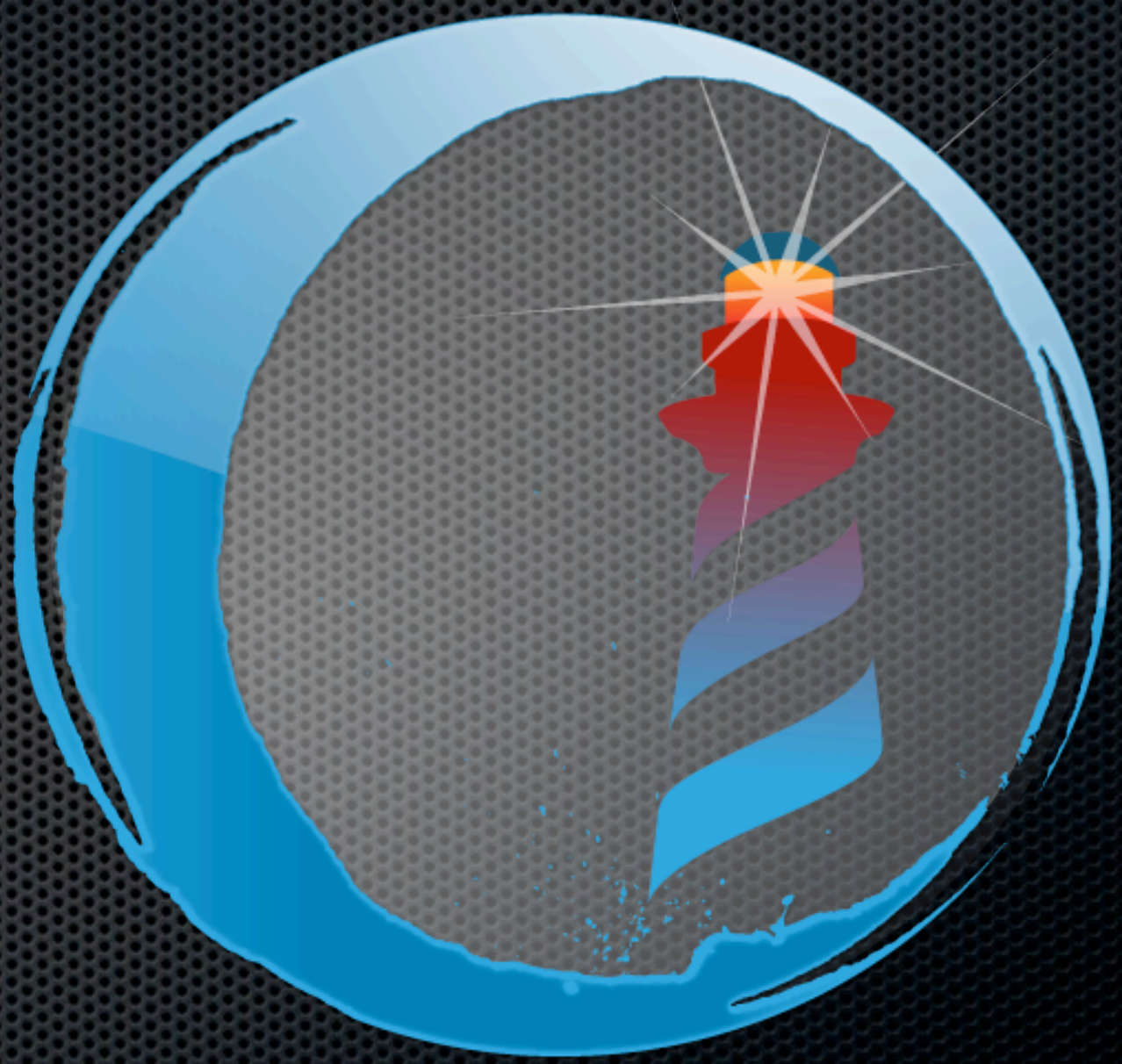
to:do:

to:by:do:

are just messages send to integers

Roadmap

Fun with iterators



```
ArrayList<String> strings
            = new ArrayList<String>();
for(Person person: persons)
        strings.add(person.name());
```

strings :=
persons collect [:person | person name].

#(2 -3 4 -35 4) **collect:** [:each| each abs]

#(2 -3 4 -35 4) **collect:** [:each| each abs]

> #(2 3 4 35 4)

#(15 10 19 68) **collect:** [:i | i odd]

#(15 10 19 68) **collect:** [:i | i odd]

> #(true false true false)

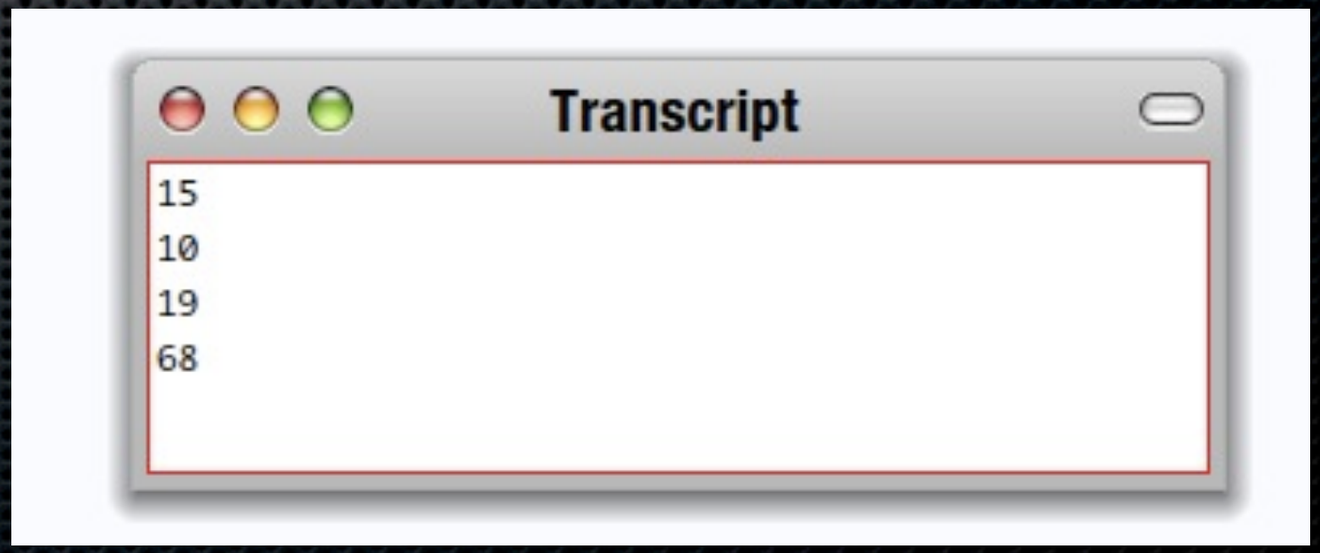
#(15 10 19 68) **collect:** [:i | i odd]

We can also do it that way!

```
|result|
aCol :=  #( 2 -3 4 -35 4).
result := aCol species new: aCol size.
1 to: aCollection size do:
        [ :each | result at: each put: (aCol at: each) odd].
result
```

```
#(15 10 19 68) do:
    [:i | Transcript show: i ; cr ]
```

```
#(15 10 19 68) do:
    [:i | Transcript show: i ; cr ]
```



#(1 2 3)

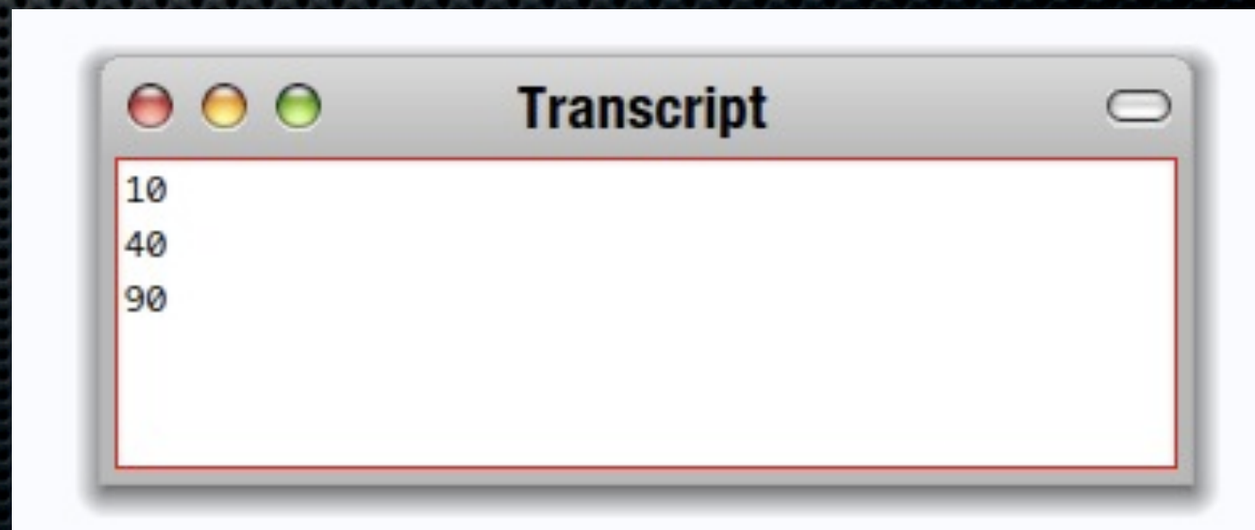
with: #(10 20 30)

do: [:x :y| Transcript show: (y ** x) ; cr]

#(1 2 3)

with: #(10 20 30)

do: [:x :y| Transcript show: (y ** x) ; cr]



How do: is implemented?

How do: is implemented?

SequenceableCollection>>do: aBlock
"Evaluate aBlock with each of the receiver's elements as the argument."

1 **to:** self size **do:** [:i | aBlock value: (self **at:** i)]

```
#(15 10 19 68) select: [:i|i odd]

#(15 10 19 68) reject: [:i|i odd]

#(12 10 19 68 21) detect: [:i|i odd]

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]
```

```
#(15 10 19 68) select: [:i|i odd]
> #(15 19)
#(15 10 19 68) reject: [:i|i odd]

#(12 10 19 68 21) detect: [:i|i odd]

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]
```

```
#(15 10 19 68) select: [:i|i odd]

> #(15 19)

#(15 10 19 68) reject: [:i|i odd]

> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]


#(12 10 12 68) detect: [:i|i odd] ifNone:[1]
```

```
#(15 10 19 68) select: [:i|i odd]

> #(15 19)

#(15 10 19 68) reject: [:i|i odd]

> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]

> 19

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]
```

```
#(15 10 19 68) select: [:i|i odd]

> #(15 19)

#(15 10 19 68) reject: [:i|i odd]

> #(10 68)

#(12 10 19 68 21) detect: [:i|i odd]

> 19

#(12 10 12 68) detect: [:i|i odd] ifNone:[1]

> 1
```

Iterators are your best friends

- compact
- nice abstraction
- Just messages sent to collections

Iterators are your best friends

compact

nice abstraction

Just messages sent to collections

How do you define the method that does that?

#() -> ‘’

#(a) ->’a’

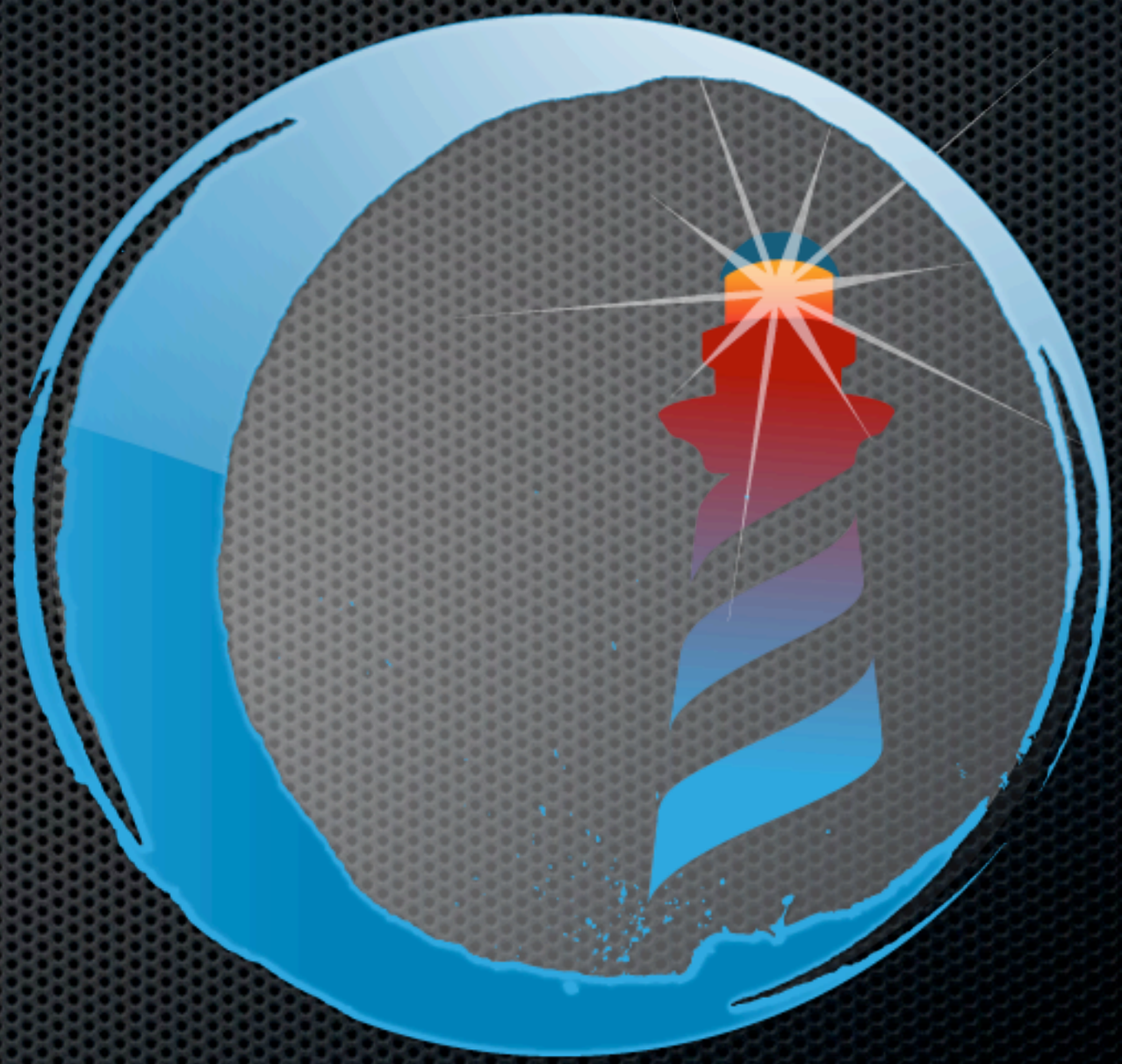
#(a b c) -> ‘a, b, c’

```
#(a b c)
    do: [:each | Transcript show: each printString]
    separatedBy: [Transcript show: ‘,’]
```

```
#(a b c)
    do: [:each | Transcript show: each printString]
    separatedBy: [Transcript show: ',']
```

Roadmap

Cascading



Messages Sequence

```
message1 .
message2 .
message3
```

. is a separator, not a terminator

```
| macNode pcNode node1 printerNode |
macNode := Workstation withName: #mac.
```

Multiple messages to an objects ;

To send multiple messages to the same object

```
Transcript show: 1 printString.
Transcript cr
```

is equivalent to:

WHAT ARE YOU DOING?! YOU'RE GOING TO BE LATE FOR SCHOOL! HURRY UP AND PUT YOUR CLOTHES ON RIGHT!

IT'S SAD HOW SOME PEOPLE CAN'T HANDLE A LITTLE VARIETY.

Pharo

http://www.pharo.org