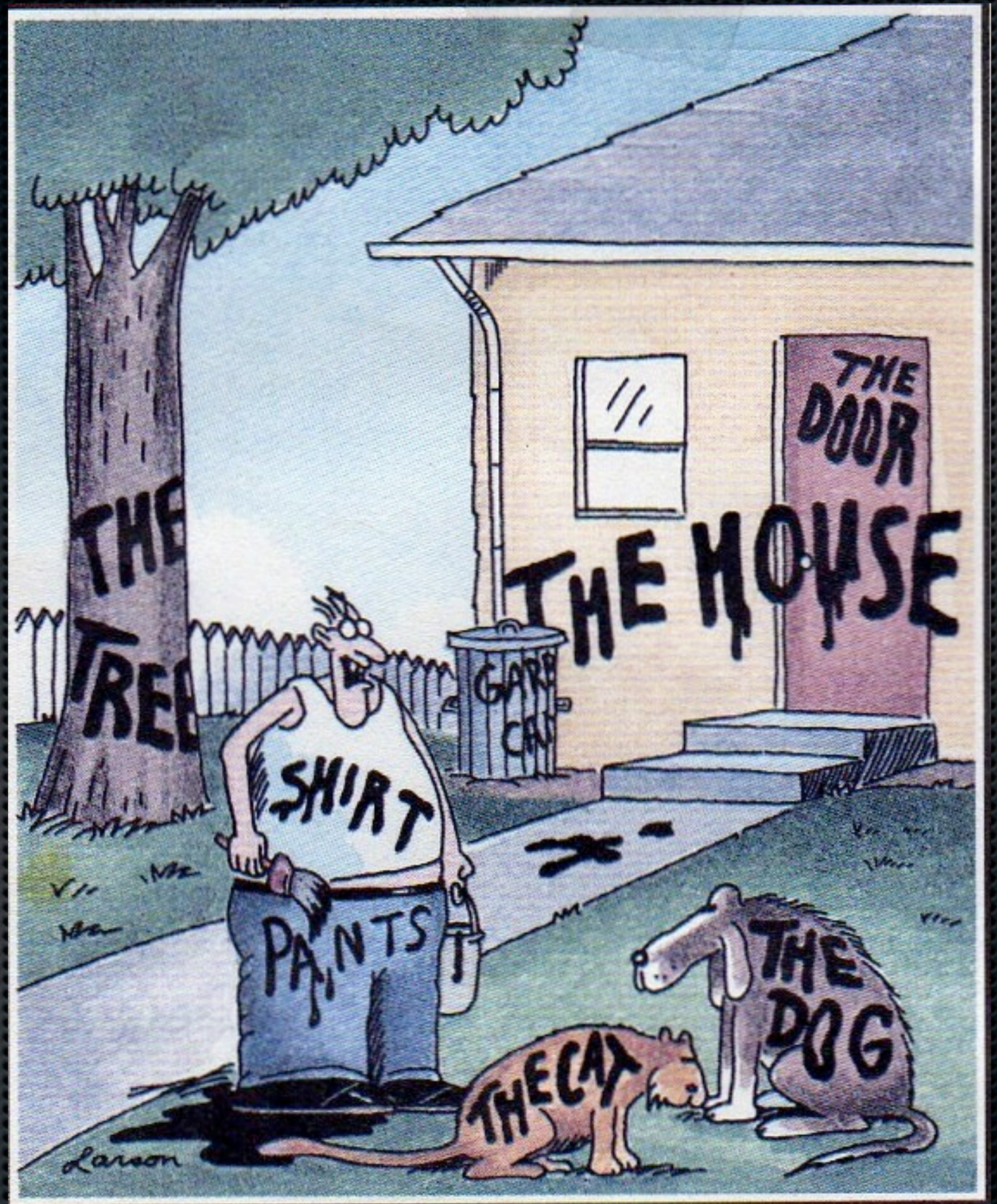
Pharo: Syntax in a Nutshell

S. Ducasse and M. Denker

http://www.pharo.org

Less is better

- No constructors
- No types declaration
- No interfaces
- No packages/private/protected
- No parametrized types
- No boxing/unboxing
- **And really powerful**



"Now! ... *That* should clear up
a few things around here!"

Everything is an object

(10@200) class

Point selectors

Class definition

Object subclass: #**Point**

instanceVariableNames: **'x y'**

classVariableNames: ''

category: 'Graphics-Primitives'

Complete Syntax on a PostCard

exampleWithNumber: x

"A method that has unary, binary, and key word messages, declares arguments and temporaries (but not block temporaries), accesses a global variable (but not and instance variable), uses literals (array, character, symbol, string, integer, float), uses the pseudo variable true false, nil, self, and super, and has sequence, assignment, return and cascade. It has both zero argument and one argument blocks."

```
|y|

true & false not & (nil isNil) ifFalse: [self halt].

y := self size + super size.

{ 1 . 2 . #($a #a 'a' 1 1.0) }

    do: [:each | Transcript show: (each class name); show: (each printString);
    show: ' '].

^ x < y
```

Language Constructs

^	return
"	comments
#	symbol or array
'	string
[ ]	block or byte array
.	separator and not terminator (or namespace access in VW)
;	cascade (sending several messages to the same instance)
|	local or block variable
:=	assignment
$	character

Syntax

comment: “a comment”

character: $c $h $a $r $a $c $t $e $r $s $# $@

string: ‘a nice string’ ‘lulu’ ‘l”idiot’

symbol: #mac #+

array: #(1 2 3 (1 3) $a 4)

byte array: #[1 2 3]

integer: 1, 2r101

real: 1.5, 6.03e-34,4, 2.4e7

float: 1/33

boolean: true, false

point: 10@120

Messages to Objects

3 kinds of messages

Unary messages

```
5 factorial
Transcript cr
```

Binary messages

```
3 + 4
```

Keywords messages

```
2 between: 0 and: 5

Transcript show: 'hello world'
```

() > Unary > Binary > Keywords

1 class maxVal raised: 3+ 2

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

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

postman send mail recipient

postman send mail to recipient

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

A typical method in Point

Method name Argument Comment

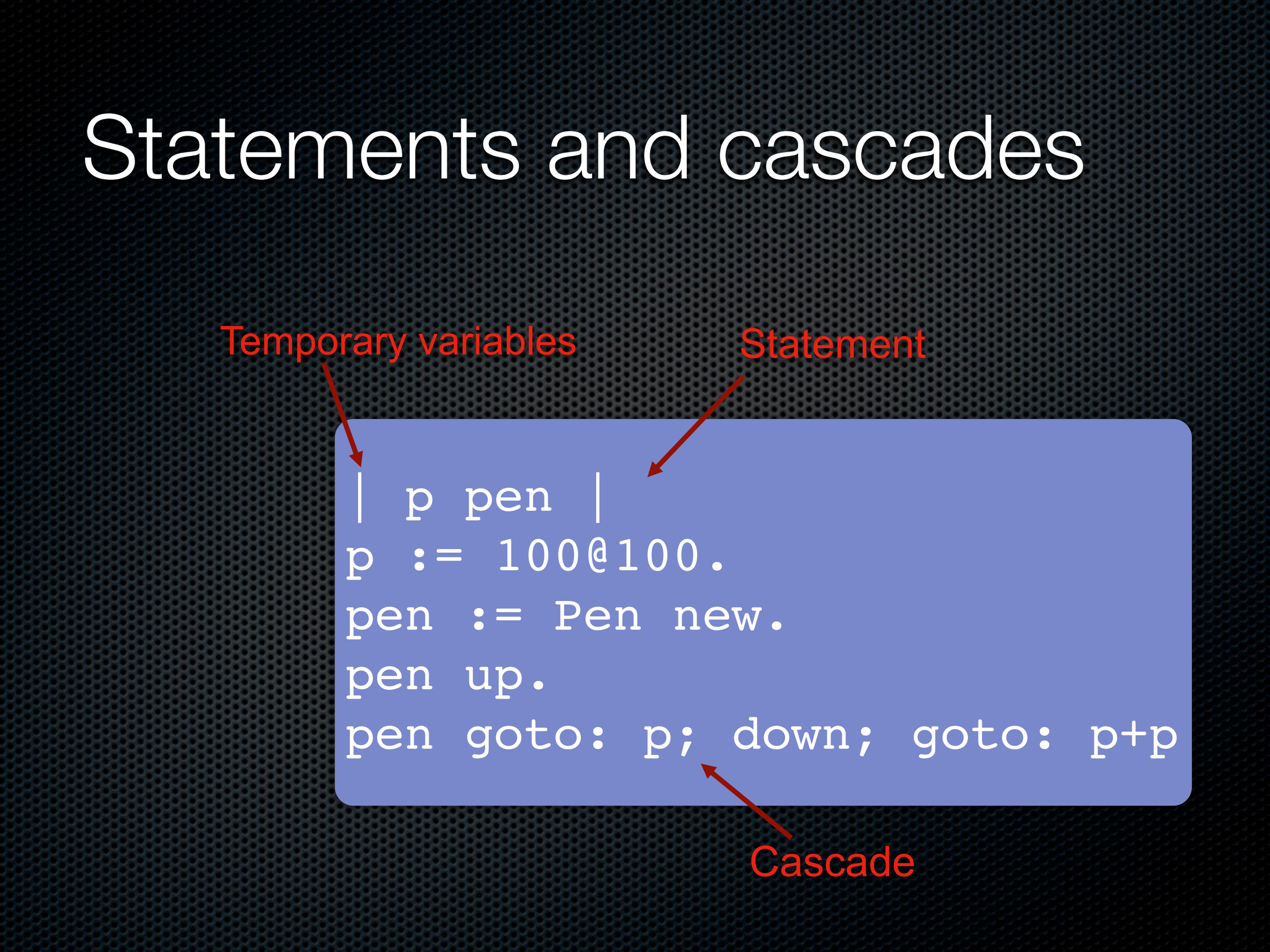
```
<= aPoint
  "Answer whether the receiver is neither
  below nor to the right of aPoint."

  ^ x <= aPoint x and: [y <= aPoint y]
```

Return Instance variable Binary message Keyword message Block

```
(2@3) <= (5@6)
```

```
true
```

Statements and cascades



Blocks

- Anonymous method
- Passed as method argument or stored

- Functions

```
fct(x)= x*x+3, fct(2).

fct := [:x| x * x + 3].
fct value: 2
```

Block usage

```
Integer>>factorial
    | tmp |
    tmp := 1.
    2 to: self do: [:i| tmp := tmp * i]


#(1 2 3) do: [:each | each crLog]
```

Control structures

Every control structure is realized by message sends

```
4 timesRepeat: [Beeper beep]
```

```
max: aNumber
  ^ self < aNumber
      ifTrue: [aNumber]
      ifFalse: [self]
```

Iterators are your best friends

compact

nice abstraction

Just messages sent to collections

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

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

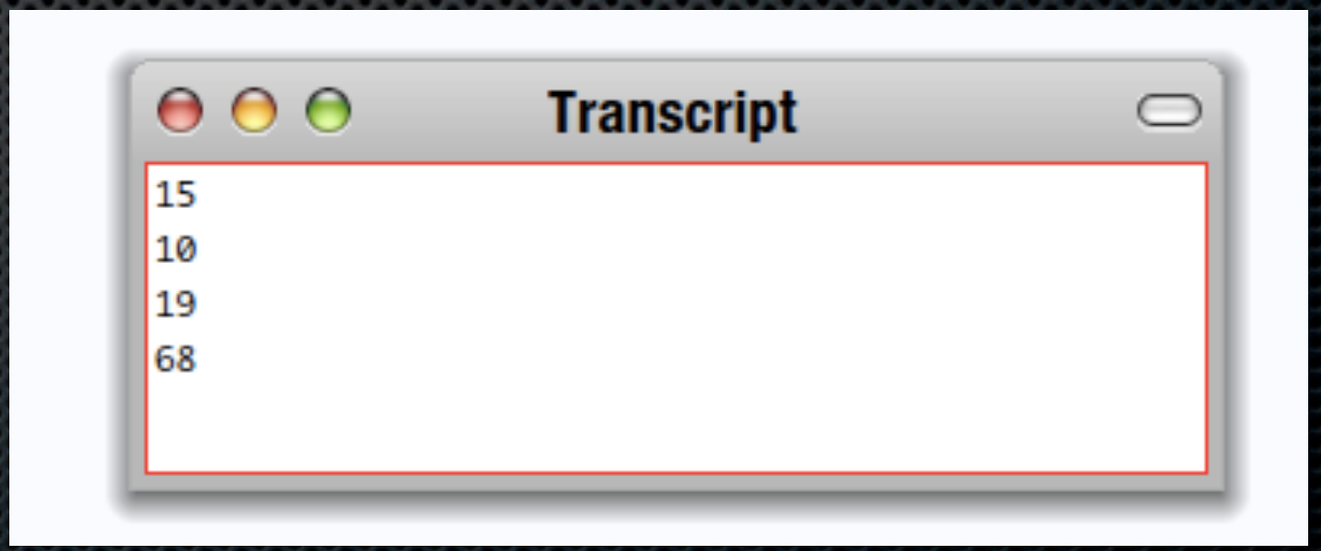
> #(2 3 4 35 4)

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

```
#(15 10 19 68) collect: [:i | i odd ]
> #(true false true false)
```

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

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

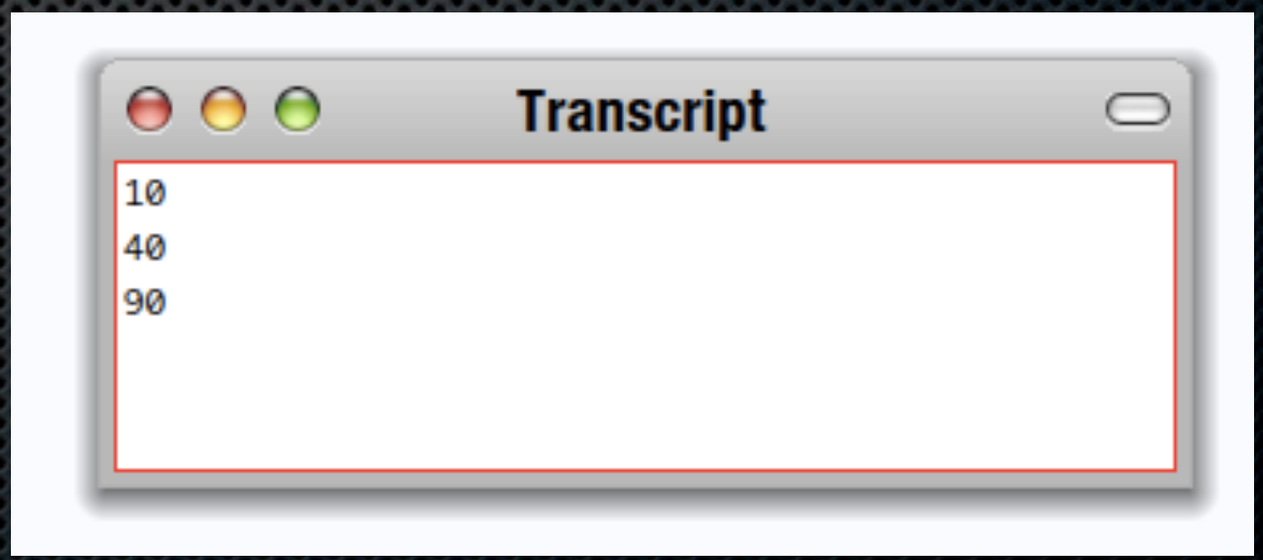


```
#(1 2 3)
    with: #(10 20 30)
    do: [:x :y| Transcript show: (y ** x) ; cr ]
```

#(1 2 3)

with: #(10 20 30)

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



```
#(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]
```

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

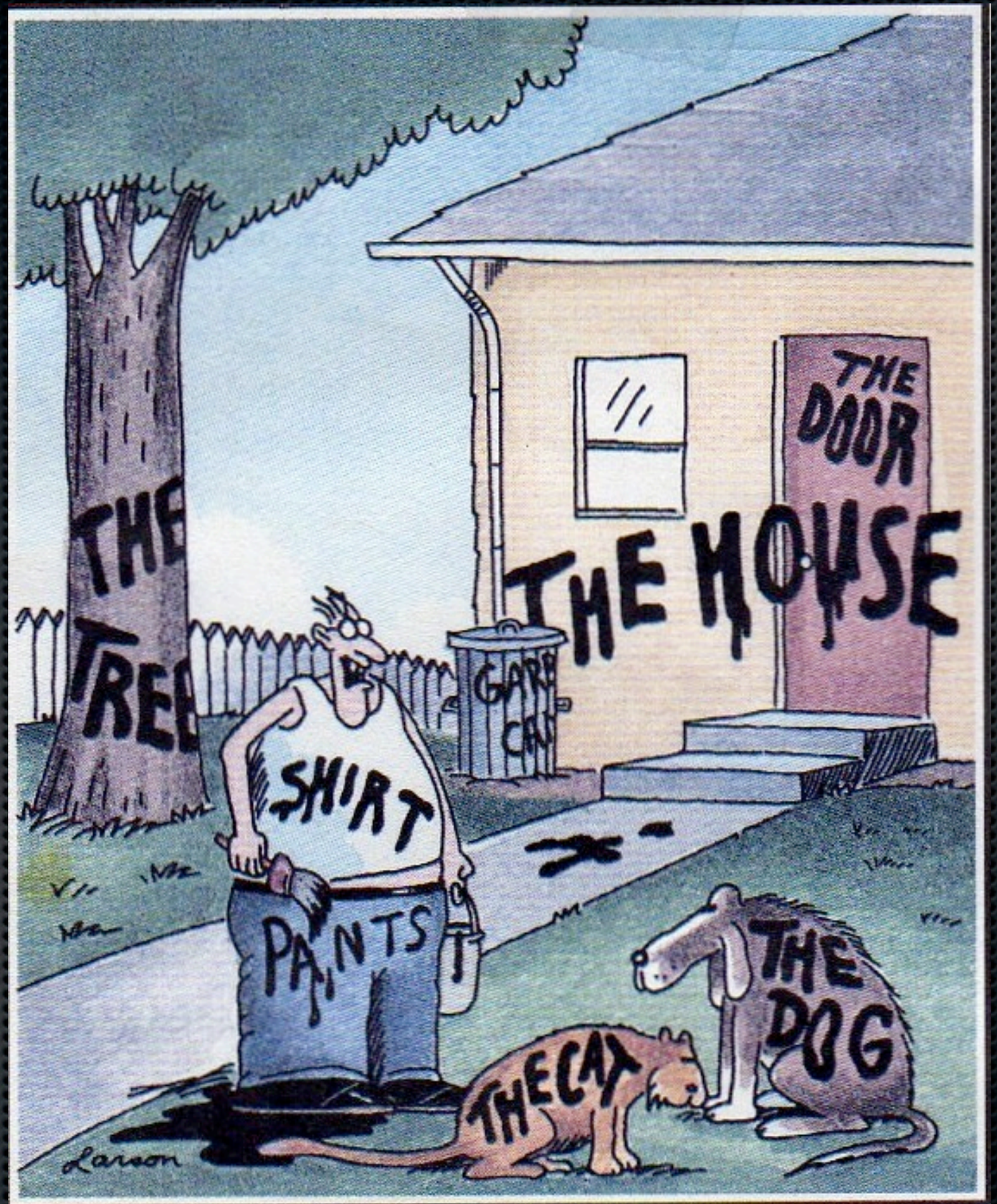
402387260077093773543702433923003985719374864210714632543799104299385123986290205920442084869694048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782973084313928444032812315586110369768013573042161687476096758713483120254785893207671691324484262361314125087802080002616831510273418279770478463586817016436502415369139828126481021309276124489635992870511496497541990934222156683257208082133318611681155361583654698404670897560290095053761647584772842188967964624494516076535340819890138544248798495995331910172335555660213945039973628075013783761530712776192684903435262520001588853514733161170210396817592151090778801939317811419454525722386554146106289218796022383897147608850627686296714667469756291123408243920816015378088989396451826324367161676217916890977991190375403127462228998800519544441428201218736174599264295658174662830295557029902432415318161721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848259012545832716822645806652676995865268227280707578139185817888965220816434834482599326604336766017699612831860788386150279465955131156552036093988180612138558600301435694527224206344631797460594682573103790084024432438465657245014402821885252470935190620929023136493273497565513958720559654228749774011413346962715422845862377387538230483865688976461927383814900140767310446640259899490222221765904339901886018566526485061799702356193897017860040811889729918311021171229845901641921068884387121855646124960798722908519296819372388642614839657382291123125024186649353143970137428531926649875337218940694281434118520158014123344828015051399694290153483077644569099073152433278288269864602789864321139083506217095002597389863554277196742822248757586765752344220207573630569498825087968928162753848863396909959826280956121450994871701244516461260379029309120889086942028510640182154399457156805941872748998094254742173582401063677404595741785160829230135358081840096996372524230560855903700624271243416909004153690105933983835777939410970027753472000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1000 factorial / 999 factorial

1000 factorial / 999 factorial

> 1000

Simple and elegant



"Now! ... *That* should clear up
a few things around here!"