

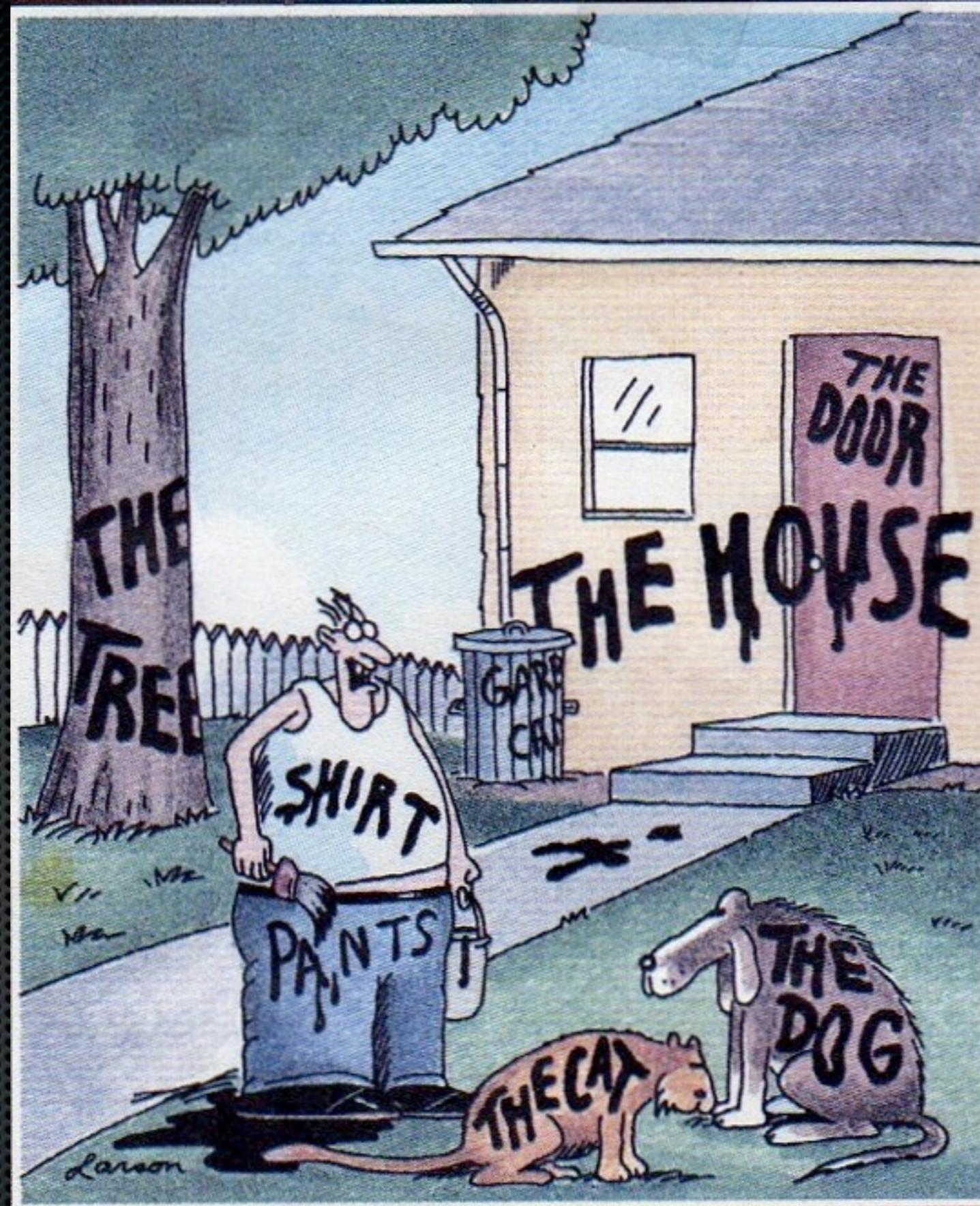
# Pharo: Syntax in a Nutshell

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."

Return Instance variable Binary message Keyword message Block

$\hat{x} \leq aPoint$   $x$  and:  $[y \leq aPoint y]$

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

true

# Statements and cascades

Temporary variables

Statement

```
| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
```

Cascade

# 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 ]
```

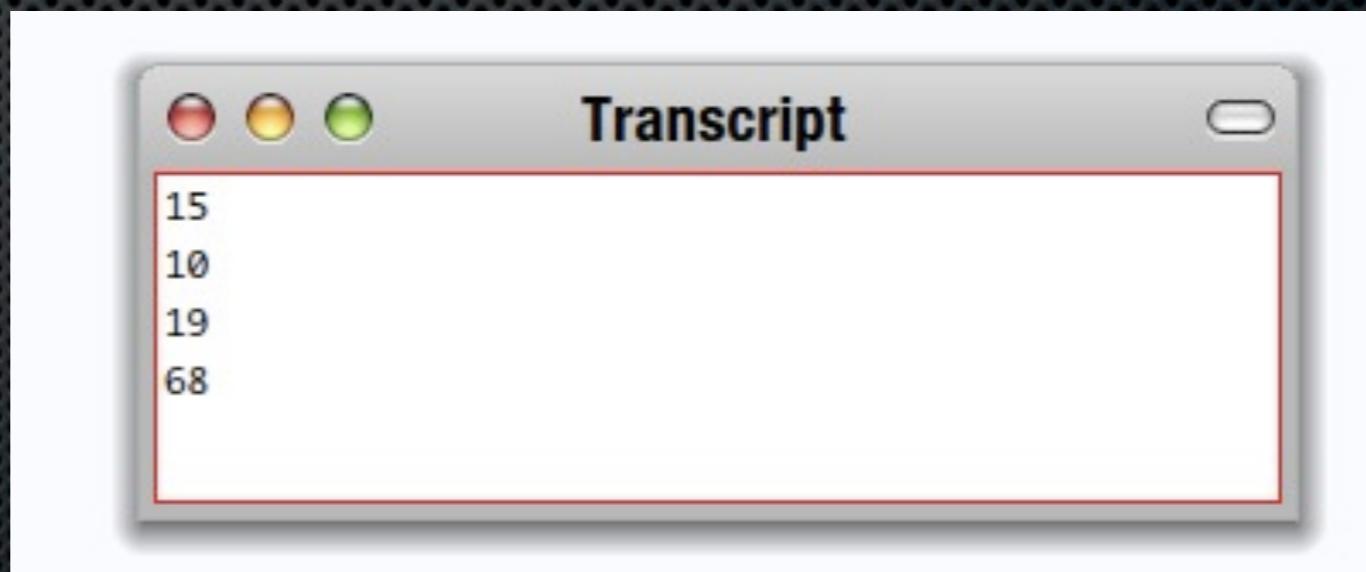
```
> #(true false true false)
```

#(15 10 19 68) **do:**

[:i | Transcript show: i ; cr ]

**#(15 10 19 68) do:**

**[:i | Transcript show: i ; cr ]**



**#(1 2 3)**

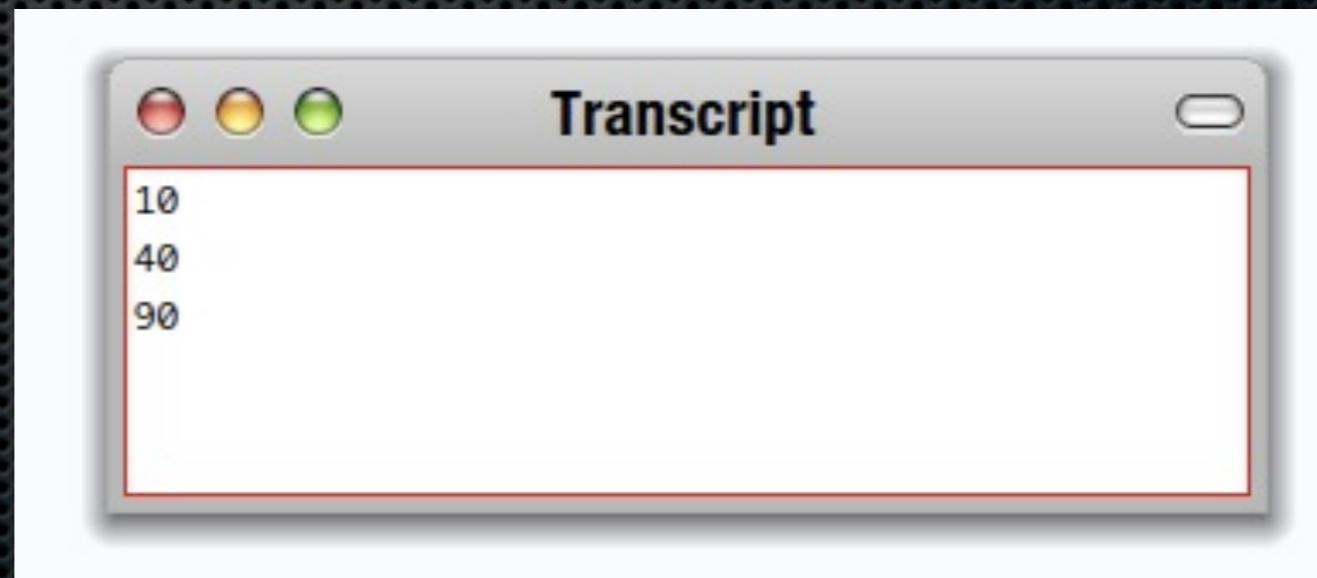
**with:** #(10 20 30)

**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

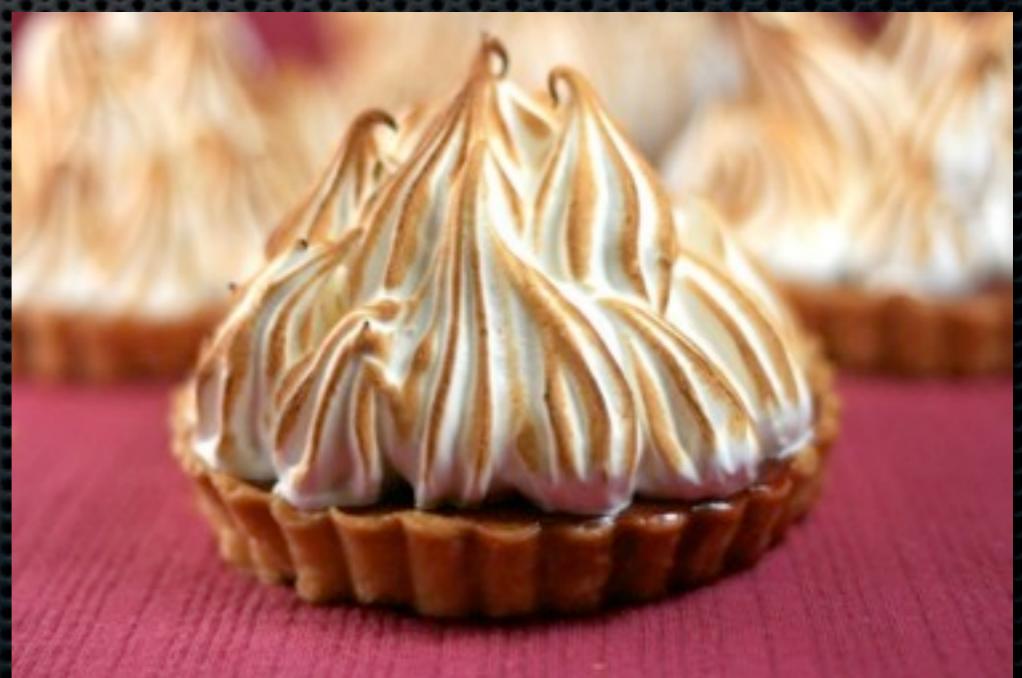
1 class maxVal + 1

>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

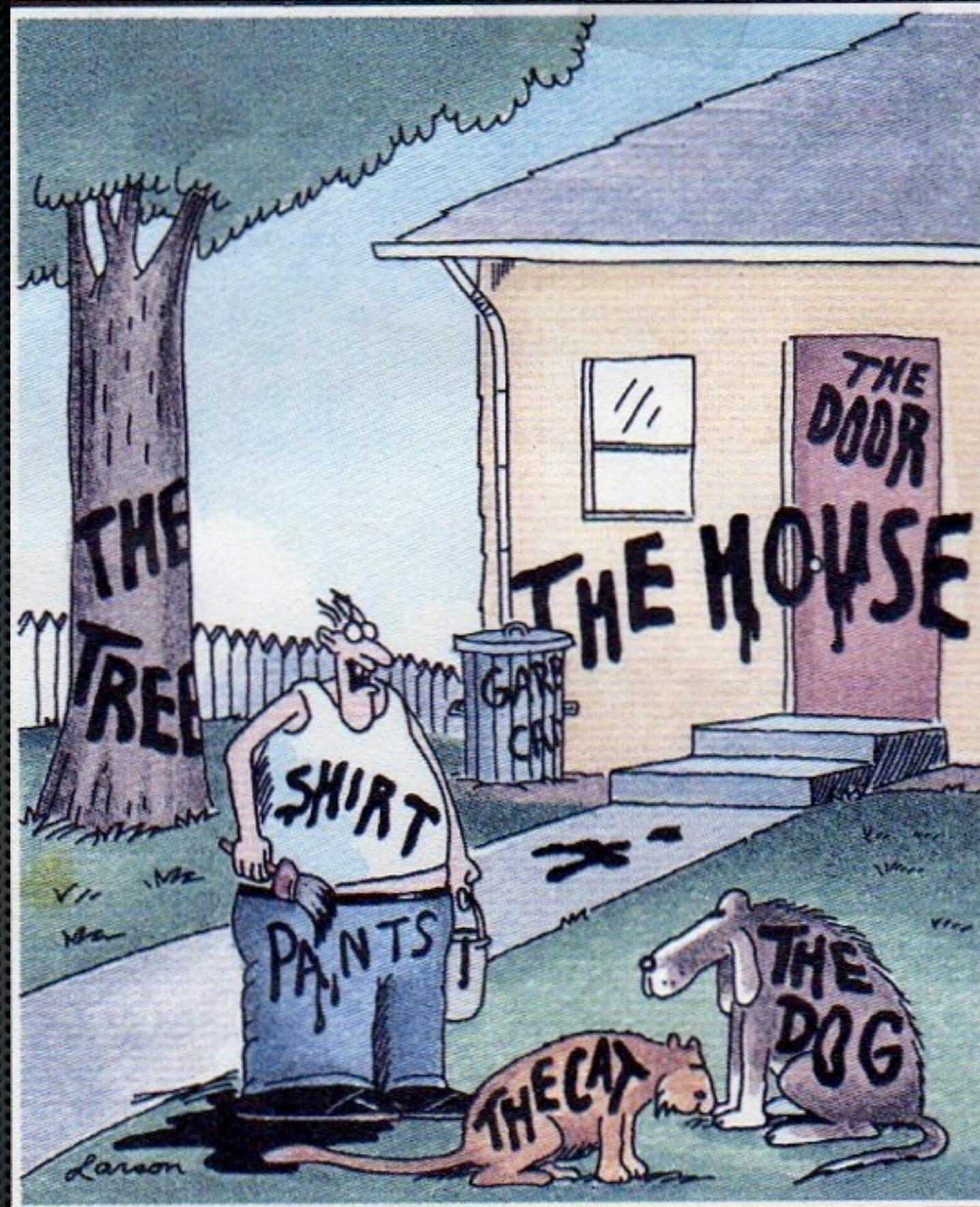
1000 factorial / 999 factorial

1000 factorial / 999 factorial

> 1000



Simple and elegant



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