



# Lisp to Ruby to Rubinius

ネットワーク応用通信研究所  
楽天 技術研究所  
Rubyアソシエーション  
@yukihiro\_matz

Yukihiro "Matz" Matsumoto

# Lisp

# Lisp



- one of the oldest
- O-Parts
  - out of place artifact

# O-Parts of the language



oldest but newest

- symbolic computation
- garbage collection
- objects
- exceptions

# Lisp

- S-expression
- macros
- everything object
- meta-programming

# Dark side of Lisp



- Parentheses
- dangling language
- there's no lisp language
- CLOS
  - powerful but complex

# Ruby



Lispy, but

- no S-expression
- no macros
- no CLOS

# Ruby



- Algol-ish syntax
- Smalltalk-ish OO
- Language for **ordinary** programmers



# Lisp

```
(defun fact (n)
  (if (= n 1)
      1
      (* n (fact (1- n)))))
(print (format "6!=~D" (fact 6)))
; => 6!=720
```

# Lisp

```
;; move parens
(defun fact (n)
  (if (= n 1)
      1
      (* n (fact (1- n)))))
)
)
(print (format "6!=~D" (fact 6)))
```

# Lisp

```
;; operator syntax
(defun fact (n)
  (if (n == 1)
      1
      (n * (fact (n - 1))))
)
)
(print (format "6!=~D" (fact 6)))
```

# Lisp

```
;; move argument parens
(defun fact (n)
  (if (n == 1)
      1
      (n * fact(n - 1)))
  )
)
print(format("6!=~D", fact(6)))
```

# Lisp

```
;; move argument parens
(defun fact (n)
  (if (n == 1)
      1
      (n * fact(n - 1)))
  )
)
print(format("6!=~D", fact(6)))
```

# Lisp

```
# syntax structures
defun fact (n)
  if (n == 1)
    1
  else
    (n * fact(n - 1))
  end
end
print(format("6!=~D", fact(6)))
```

# Lisp

```
# reduce parens
defun fact (n)
  if n == 1
    1
  else
    n * fact(n - 1)
  end
end
print(format("6!=~D", fact(6)))
```

# Ruby

```
def fact (n)
  if n == 0
    1
  else
    n * fact(n - 1)
  end
end
printf "6!=%d", fact(6), "\n"
# => 6!=720
```



# Lisp vs Ruby



## Syntax

- Less parentheses
- Many 'end's
- special forms vs syntax structures

# Lisp vs Ruby



## Semantics

- Quite similar
- nearly one to one translation
- auto conversion from fixnums to bignums
  - fact 200



# MatzLisp

# MatzLisp



- not MacLisp
- not FranzLisp

# Ruby

- Lisp without S-expression
- sprinkled with syntax sugar
- with OO from Smalltalk
- operators from C
- strings/regexp from Perl

# Lisp without S-expr



Remember M-expression

# Ruby = Weak Lisp?



- No S-expression
- No Macro
- Who cares.



# Language Power $\neq$ Programming Power



# Personal History



BASIC to Lisp

# BASIC



# BASIC



# Got tired of BASIC



- No user defined functions
- No user defined data types

# Aristocracy



Language Designers  
Implementers

---

Programmers

# I met Lisp



- in an AI book
- Lisp made my eyes open
- users can do everything

# Users can do Everything



- define functions
- define data types
- enhance the language

# Democracy



- no discrimination
- users can be language implementers



# But, wait

Unlike politics (or like politics)

- freedom comes with responsibility
- ordinary people hate (too much) responsibility
- or too much power

# Too much power



- smart people love power
- smart people underestimate ordinariness of ordinary people



中庸

Happy Medium

# Happy Medium

There should be somewhere  
in between language  
aristocracy and democracy,  
where ordinary people can  
live happily, without feeling  
fear.

# Balance

It's quite easy to pursue extreme, but seeking 'something in-between' is far more difficult.



# Ruby

# Ruby



My answer to the ultimate question.

# Result

Position Sep 2010	Position Sep 2009	Delta in Position	Programming Language	Ratings Sep 2010	Delta Sep 2009	Status
1	1	=	Java	17.915%	-1.47%	A
2	2	=	C	17.147%	+0.29%	A
3	4	↑	C++	9.812%	-0.18%	A
4	3	↓	PHP	8.370%	-1.79%	A
5	5	=	(Visual) Basic	5.797%	-3.40%	A
6	7	↑	C#	5.016%	+0.83%	A
7	8	↑	Python	4.583%	+0.65%	A
8	18	↑↑↑↑↑↑↑↑↑↑	Objective-C	3.368%	+2.78%	A
9	6	↓↓↓	Perl	2.447%	-2.08%	A
10	10	=	Ruby	1.907%	-0.47%	A
11	9	↓↓	JavaScript	1.665%	-1.33%	A
12	11	↓	Delphi	1.585%	-0.39%	A
13	13	=	Lisp	1.084%	+0.24%	A--
14	12	↓↓	Pascal	0.790%	-0.17%	A--
15	27	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	0.771%	+0.40%	A--
16	-	↑↑↑↑↑↑↑↑↑↑	Go	0.728%	+0.73%	A--
17	21	↑↑↑↑	RPG (OS/400)	0.715%	+0.26%	A--
18	30	↑↑↑↑↑↑↑↑↑↑	PowerShell	0.686%	+0.42%	B
19	24	↑↑↑↑↑	Ada	0.676%	+0.29%	B
20	14	↓↓↓↓↓	PL/SQL	0.637%	-0.18%	A-



# Self-sustaining languages

- C
- Lisp
- MFTL
- Ruby

# My Favorite Toy Language



n. Describes a language about which the developers are passionate but no one else cares about.

--Jargon File

# My Favorite Toy Language



The first great goal in the mind of the designer of an MFTL is usually to write a compiler for it

--Jargon File

# I am not a fan of meta-circular implementation



# Switching the brain



- C → Core mode
- Ruby → App mode

# Self-sustaining systems



C

- Compiler written in C
- Compiled code does not rely on the language

# Self-sustaining systems



## Squeak

- Bootstrap
- Core written in Slang
  - subset of Smalltalk
  - compiles to C
- libraries in Smalltalk

# Rubinius



Ruby implementation  
influenced by Smalltalk



# Rubinius



- small VM in C++
- libraries written in Ruby

# Bootstrap

1. VM
2. alpha
3. bootstrap
4. platform
5. common
6. delta

# VM

The virtual machine is able to load and execute bytecode, send messages (i.e. look up and execute methods), and all primitive functions are available, but not yet hooked up as Ruby methods.

# VM

At this point there is enough defined behavior to begin to load up the rest of the runtime kernel which is all defined in ruby. This has to be done in several passes as the language grows.



# alpha

This starts the loading of Ruby code. The ability to open classes and modules and define methods exists. Also, it is possible to raise exceptions and cause the running process to exit. This stage lays the foundation for the next two stages.

# bootstrap

This stage continues to add the minimum functionality to support loading platform and common. The primitive functions are added for most of the kernel classes.



# platform

The FFI system is implemented and Ruby method interfaces to platform-specific functions are created. Once this is set up, platform specific things such as file access, math, and POSIX commands are attached.

# common

The vast majority of the Ruby core library classes are implemented. The Ruby core classes are kept as implementation-neutral as possible.



# delta

Implementation-specific versions of methods that override the versions provided in common are added.

# Problems



- performance
- open class

# (Possible) Solution



- JIT (LLVM)
- selector namespace / classbox



# Thank you!