Lisp (what ITA might have that we don't)

Andrey Kotlarski

28.VI.2011

Outline

History and stuff

- One of the oldest programming languages still in use
- Actually a family of languages
- Academic wing (Scheme), industrial wing (Common Lisp, maybe Clojure)
- Starting in days of limited hardware, it's quite efficient
- Accidentally or not, raised by the Artificial Intelligence pioneers and for long time being the standard there
- Lots of groundbreaking ideas for its time, most of them have slowly crept to mainstream and are now taken for granted (garbage collection, dynamic typing, tree data structures, interactive development)
- Easy to implement, has standards, lots of realisations
- Based on Alonzo Church's lambda calculus

Common Lisp nowadays

- Never quite in the mainstream, but with somewhat growing interest in recent years
- Aging standard but the language doesn't feel handicapped
- HyperSpec, excellent documentation
- Lots of implementations, 2 of them commercial
- The most popular open source implementation SBCL (Steel Bank Common Lisp) is actually the fastest and written in... Common Lisp
- Small community but with increasingly better library support
- Lots of great books
- Known usage includes 3D graphics suits, game engines, semantic web reasoning systems, knowledge and rule based systems, theorem provers, compilers, algebra systems, telecom systems, fare search engine...

Technical features

- Multi-paradigm: supports procedural, functional and object-oriented styles out of the box
- Minimal, consistent syntax based on S-expressions
- Code is data
- Programmer has essentially everything that the language creators have had, great extensibility
- Dynamic typing with optional type annotation
- Read Eval Print Loop
- Supports incremental development
- Efficiently compiled
- Macros
- Condition system
- CLOS

S-expressions and evaluation

- S-expressions (lists) are actually the abstract syntax tree that directly feeds the lisp compiler
- Each S-expression returns a value
- Evaluating non empty list normally asks the environment for the function/macro represented by the first symbol.
- When function, rest of the list is treated like arguments that are also evaluated and passed to the function.

```
(some-function arguments that are first evaluated)
```

- When macro, rest of the list is treated like arguments that are passed as they are to the macro.

```
(some-macro arguments passed as they are)
```

- Lists are treated as function/macro invocations unless quoted

```
'(some list with unevaluated elements)
```

Variables

- Lexical scope by default, with a twist

Example (Closures)

```
(let ((counter 0))
  (defun inc-counter ()
    (incf counter))

  (defun dec-counter ()
    (decf counter)))
```

Variables (continued)

Example (Dynamic aka special variables)

```
(defparameter *debug* nil)

(defun bla-bla ()
  (no-debugging)
  (let ((*debug* t))
    (do-some-stuff-with-debugging))
  (no-debugging))
```

Functions

- First class citizens
- Anonymous functions
- Functions as data
- Multiple return values

Macros

- Program life-cycle
- Run-time vs. compilation
- Macro expansion time
- Programming the compiler
- Almost like functions on the outside
- Programming over the source code with all the power of the language

Condition System

- Beyond exception handling
- Conditions and restarts
- Condition handlers

Example (handler-case similar to catch)

```
(handler-case
    (progn
      (do-stuff)
      (do-more-stuff))
  (some-exception (se) (recover se)))
```

Condition System (continued)

Example (restart-case)

```
(defun parse-log-file (file)
  (with-open-file (in file :direction :input)
    (loop for text = (read-line in nil nil)
          while text
          for entry = (restart-case
                          (parse-log-entry text)
                        (skip-log-entry () nil))
          when entry collect it)))
```

Condition System (continued)

Example (handler-bind)

```
(defun log-analyzer ()
  (handler-bind
      ((malformed-log-entry-error
         #'(lambda (c)
             (invoke-restart 'skip-log-entry))))
    (dolist (log (find-all-logs))
      (analyze-log log))))
```

- Signals, why just errors
- Restarts at lower levels, handlers at higher

Common Lisp Object System

- Message passing
- Decoupling classes from methods
- Generic functions
- Method combinations
- Multimethods

Switch to Emacs please

...

Books

- Practical Common Lisp
- HyperSpec
- Common Lisp the Language, 2nd Edition
- On Lisp: Advanced Techniques for Common Lisp
- Common Lisp: A Gentle Introduction to Symbolic Computation
- Paradigms of Artificial Intelligence Programming
- Structure and Interpretation of Computer Programs

Links

- CLiki
- CL resources
- Implementations: A Survey
- Quicklisp - CL package manager
- SLIME: The Superior Lisp Interaction Mode for Emacs
- Dr. Edmund Weitz's great libraries
- Steel Bank Common Lisp
- Some SBCL benchmarks
- Franz Inc.
- LispWorks

Land of Lisp- The Music Video!

Figure: Secret alien technology

Figure: Lispers

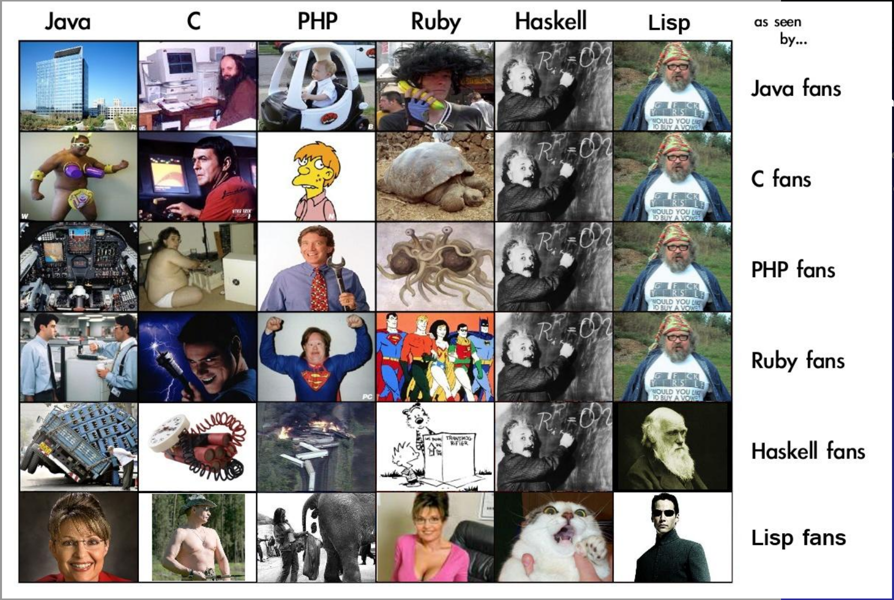

Figure: Fanboys