Common Lisp の紹介

biyori-sh (2021/02)

イントロ前: Common lisp の見た目

巷の噂では…

- 括弧が多い
  ⟶ 後ろの括弧はまとめる習慣があるのが影響？
- ポーランド記法
  ⟶ そのとおり、基本的に演算子、引数の順
- 関数型言語
  ⟶ そうとも言えるしそうでないとも言える

```
(+ 1 2 3) ; => 6
(* 1 2 3) ; => 6
(/ 6 3 2) ; => 1
```

```
(defun fib (n)
  "simple␣implementation␣using␣recursion"
  (cond ((<= n 0) 0)
    ((= n 1) 1)
    (t (+ (fib (- n 1))
          (fib (- n 2))))))

(fib 5) ; => 120
```

見た目がメジャーどころの言語とちょっと違うマルチパラダイム言語

イントロ前: どうして Common Lisp?

特に強い理由は無い

- 言語そのものの”””” 利便性”””” はとくに無い…が
- 言語そのものが面白い、そして
- 言うほど不便でもない（書きやすさ・実行速度）

個人的には… Common Lisp は微妙に低級っぽさがあり、勉強になる（気がする）

Fortran, C/C++ っぽくない言語ならどれでもいいのかもしれない、とは言え Julia は

※ Common lisp の強みを述べる上でマクロは欠かせないと思われるが、
実力不足ゆえに殆ど扱えないので、主に初歩の初歩的なことを扱います。
あとクロージャーにも触れてません。
Common lisp object system (CLOS) も全然わかってないので扱えません。

イントロ: Common Lisp (LISP) の歴史

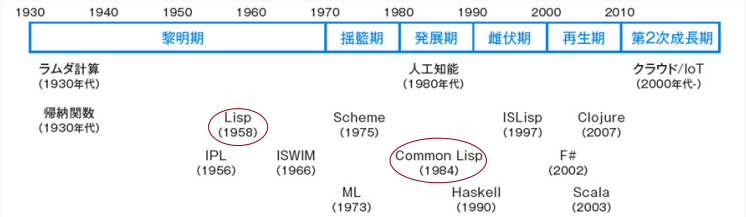


図1.4▶関数型プログラミングの歴史（Lispの歴史を含む）

引用: 五味弘『はじめての Lisp 関数型プログラミング』より抜粋（一部改変） ← この色リンク

1958 年: John McCarthy らによって LISP 誕生（FORTRAN が 1957 年、COBOL が 1960 年）

プログラミング言語年表-Wikipedia

1972 年に Dennis Ritchie らが C 言語、1990 年に Guido van Rossum によって Python が誕生

1975 年: Scheme

1984 年: Common Lisp ⟶ 1994 年: Ansi Common Lisp

Scheme や Common Lisp は LISP の方言の一種

イントロ: 処理系

ANSI で定められた仕様を満たしさえすれば Common Lisp を名乗れるので
複数の実装がある。

- CLISP: 気楽に使いやすい実装。インタプリタだったりコンパイラだったり。
- SBCL: おそらく 1 番使われてる実装。定義した函数は一度コンパイルされて以降は実行される。
- Allegro Common Lisp: 商用の実装。

など

イントロ: LISP といえば

- なんと言っても S 式 ⟵ リスト構造で記述された形式
- 初めてガベージ・コレクション (GC) を実装した言語
- 動的型付け言語
- 函数が第 1 級オブジェクト
- S 式
- インタプリタ言語（と言って良いのかな？） #:g1: SBCL のインタプリタが強化されたらしい
- 一昔前の人工知能の研究でよく使われていたらしい言語
- やっぱり S 式

Common Lisp の特徴: S 式

S 式（エスしき、英: *S-expression*）とは、*Lisp* で導入され、主に *Lisp* で用いられる、コンスセル、二分木ないしリスト構造の形式的な記述方式。*S* は *Symbol* に由来。

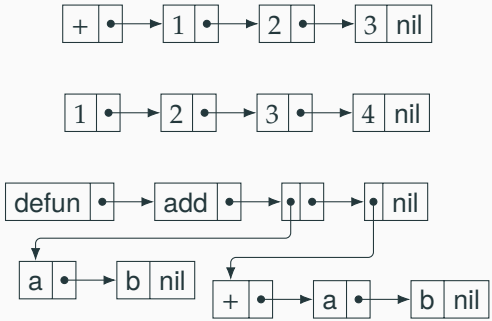
S 式-Wikipedia

```
> (+ 1 2 3)          ; => 6, S-expression

> '(1 2 3 4)         ; list, true
> '()                ; list, false
> 'a                  ; atom (symbol), true
> 7                  ; atom (number), true
> t                  ; atom (symbol), true
> nil                ; atom, false
```

`'a` は `(quote a)` の糖衣構文

```
> (cons 1 '(2 3)) ; => (1 2 3)
> (car '(1 2 3))  ; => 1
> (cdr '(1 2 3))  ; => (2 3)

> (defun add (a b) (+ a b)) ; define a function
```

二分木構造になっている。
リストとそれ以外のアトムとで構成される。



Common Lisp の特徴: S 式（よく使うものとか）

```
> (defparameter a 21)      ; a = 21 in python (global)
> (setq b 32)              ; b = 32 in python (global), Not good manner

> (let ((a 1)              ; local variables a and c
        (c 2))
      (setq a (+ a c))
      (print a))           ; => 3, not 23
> c                        ; => "The variable C is unbound."

> (labels ((my-add (a b) (+ a b)))
            (my-add 1 2)); => 3
> (my-add 1 2)             ; => "The function COMMON-LISP-USER::ADD is undefined."

> (funcall #'sin (/ pi 2)); => 1.0, sin(pi/2) in python

> (lambda (x) (* x x))     ; => #<FUNCTION (LAMBDA (X)) {53062ACB}>
> (mapcar #'(lambda (x) (* x x))
          '(1 2 3 4 5))    ; => (1 4 9 16 25)
```

Common Lisp の特徴: 評価 1/2

Common lisp、あるいは Lisp 特有というわけでは全く無いが、基本なので…

Read–Eval–Print–Loop (REPL)

「S 式を読み取って評価（実行）し、その返り値を出力」を繰り返す

`(+ 5 2 (/ 3 1.5))`（いわゆる `5 + 2 + 3/1.5 = 9.0`）を例に示す。

- `+` を読み取る。先頭なので函数として読み取る。（まだ評価されない。）
- 左から順に読み取り、評価する。`5` -> `5` と `2` -> `2`。（引数を先に評価する。）
- `(/ 3 1.5)` を読み取る。`/` を函数として読み取る。同様に `3` -> `3` と `1.5` -> `1.5`。
- `(/ 3 1.5)` の引数を評価し終わったので、`(/ 3 1.5)` を評価する。`(/ 3 1.5)` -> `2.0`。
- 引数の評価が終わったので `(+ 5 2 2.0)` を評価する。`(+ 5 2 2.0)` -> `9.0`

基本的に引数を先に評価する（正格評価）。また式なので必ず返り値がある。

c.f. 文 （いわゆる for 文、if 文） -> 例外: マクロ、特殊式

Common Lisp の特徴: 評価 2/2

特殊式: `if` や `while`、 `and`、 `quate` など

```
> (if (= 0 1)
      (/ 1 0)
      (format t "zero␣is␣not␣one!"))
> zero is not one!       ; standard output
> NIL                    ; return value

> (or t (/ 1 0))
> T                      ; return value

> (and nil (/ 1 0))
> NIL                    ; return value
```

引数に含まれる `(/ 1 0)` は評価されればエラーが出てプログラムは止まる。

左の例では評価されていないのでエラーは表示されない。

Common Lisp の特徴: 同図像性

- 函数で扱うデータ構造としてのリスト
- データを扱う函数（あるいは言語の構文）そのもののデータ構造がリスト

つまり 扱うもの と 扱われるもの とが 同じ構造（リスト） になっている

メタプログラミングとの親和性 → マクロ

コードを生成するコード

- 通常の式: 引数を先に評価して、式を評価する。
- マクロ: 評価すると指定した引数、あるいはコード以外の評価を保留して S 式を生成する（展開）。そのあと生成した S 式を評価する。

Common Lisp の特徴: マクロ 1/3

使うタイミングとしては悪い例、だけど自分が書いたのがこれしか無かったので…

本当はメモ化の自動化とか構文の変更などで使うようなもの

```
> (defmacro generalized-fizzbuzz (n &rest pairs-div-str)
    "generalized␣implementaion␣based␣on␣fizzbuzz-strict-nor"
    (let ((cnt (gensym))
          (lst-if nil))
      (dolist (lst (sort pairs-div-str #'> :key #'car))
        (push `(if (= (mod ,cnt ,(car lst)) 0) (princ ,(cadr lst)))
              lst-if))
      `(labels ((princ-strict-nor (k &rest tests)         ;;
                  (when (every #'null tests) (princ k)))) ;;
         (do ((,cnt 1 (1+ ,cnt)))                         ;;
             ((> ,cnt ,n))                                ;;
           (princ-strict-nor ,cnt ,@lst-if)               ;;
           (format t "~%")))))                            ;;
> GENERALIZED-FIZZBUZZ
```

Common Lisp の特徴: マクロ 2/3

使うタイミングとしては悪い例、だけど自分が書いたのがこれしか無かったので…

本当はメモ化の自動化とか構文の変更などで使うようなもの

```
> (generalized-fizzbuzz 11 (2 "fizz") (3 "buzz") (5 "foo"))
> 1
  fizz
  buzz
  fizz
  foo
  fizzbuzz
  7
  fizz
  buzz
  fizzfoo
  11                              ; standard output
> NIL                             ; return value
```

Common Lisp の特徴: マクロ 3/3

使うタイミングとしては悪い例、だけど自分が書いたのがこれしか無かったので…

本当はメモ化の自動化とか構文の変更などで使うようなもの

```
> (macroexpand '(generalized-fizzbuzz 11 (2 "fizz") (3 "buzz") (5 "foo")))
> (LABELS ((PRINC-STRICT-NOR (K &REST TESTS)
             (WHEN (EVERY #'NULL TESTS) (PRINC K))))
    (DO ((#:G649 1 (1+ #:G649)))
        ((> #:G649 11))
      (PRINC-STRICT-NOR #:G649
       (IF (= (MOD #:G649 2) 0)
           (PRINC "fizz"))
       (IF (= (MOD #:G649 3) 0)
           (PRINC "buzz"))
       (IF (= (MOD #:G649 5) 0)
           (PRINC "foo")))
      (FORMAT T "~%")))          ; standard output
> T                              ; return value
```

パッケージ

Quicklisp を使えば Common lisp の内でパッケージをインストール・管理が出来る。

たとえば GNU Scientigic Library (GSL) でガンマ函数 $\Gamma(s) = \int_0^\infty \mathrm{d}x\, x^{s-1}e^{-x}$

```
> (ql:quickload :gsll) ; => (:gsll)
> (setq *read-default-float-format* 'double-float) ; => DOUBLE-FLOAT

> (gsl:gamma 2.1)                ; gamma function
  1.046485846853562              ; value
  7.002670722512676e-15          ; error

> (gsl:integration-QAGiu         ; semi-infinite interval integral
     (lambda (x) (* (expt x (- 2.1 1.0)) (exp (* -1.0 x)))) 0.0)
  1.046485842792686              ; value
  2.7015091459371097e-6          ; error
```

補足: Common lisp はデフォルトで多値函数が実装されている。

締めみたいな何か

Python とかFortran, C/C++ 以外にも色んな言語があるぞ！

何が出来るとか気にしないで触ってみるのも面白いかもよ

というか特定の言語でしか出来ないことのほうが稀では…

Julia, OCaml, Haskell, Rust,...