Lispは関数型言語（ではない）

Lisp is (not) a functional language

pixiv Inc.
USAMI Kenta

pixiv

2025-06-15 #fp_matsuri
関数型まつり 2025

お前誰よ

- うさみけんた（@tadsan）/ **Zonu.EXE** / **にゃんだーすわん**
- ピクシブ株式会社 pixiv事業本部 Webエンジニアリングチーム PHPer
  - 2012年末から現職、APIとかCIとかいろいろなところを見つめてきました
  - 最近はピクシブ百科事典(dic.pixiv.net)も開発しています
- Emacs PHP Modeを開発しています（2017年-）
- プログラミング言語にちょっとこだわりのある素人（spcamp2010）

emacs-php

Type / to search

Overview | Repositories 39 | Projects | Packages | Teams 1 | People 10 | Insights | Settings

Friends of Emacs-PHP development

Join us!

14 followers | <?php | https://www.emacs-php.dev/

Pinned

Customize pins

php-ts-mode Public

A Tree-sitter based major mode for editing PHP codes

Emacs Lisp | 23 | 8

php-mode Public

A powerful and flexible Emacs major mode for editing PHP scripts

Emacs Lisp | 598 | 119

phpactor.el Public

Interface to Phpactor (an intelligent code-completion and refactoring tool for PHP)

Emacs Lisp | 43 | 12

phpstan.el Public

Interface to PHPStan (PHP static analyzer)

Emacs Lisp | 26 | 15

Repositories

pixiv

PHPStan

Home | Try | Documentation | Blog | Search ⌘ K

Meet The Next Member of Your Team!

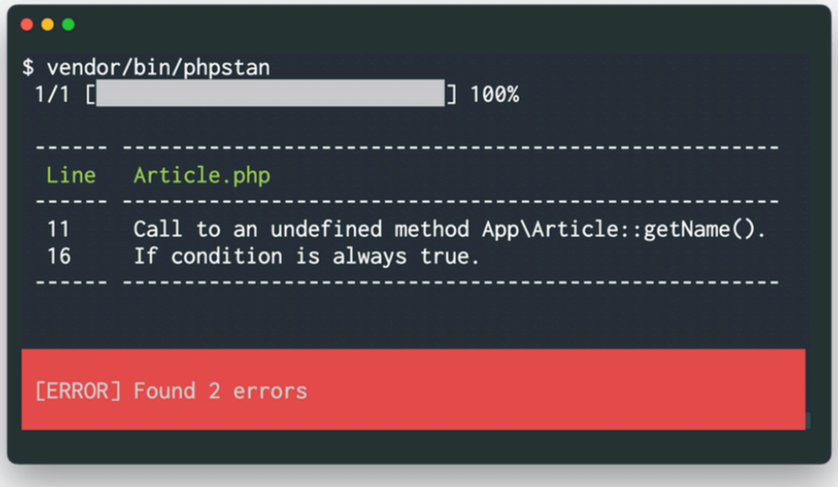
PHPStan finds bugs in your code without writing tests. It's open-source and free.

Get Started | Try It Online

Find bugs before they reach production

PHPStan scans your whole codebase and looks for both obvious & tricky bugs. Even in those rarely executed if statements that certainly aren't covered by tests.

You can run it on your machine and in CI to prevent those bugs ever reaching your customers in production.



pixiv

PHPStan型付けマニュアル

2025/03/30に公開 2025/04/15

PHP PHPStan Tech

こんにちは！ 楽しくPHPStanを使っていますか？ それともPHPStanに使われていますか？

PHPStanは非常に賢く、容易にPHPコードの *「嫌な気配」* を検知してくれます。ただ、PHPStanが指摘することが常に正しいなどといったことはなく、いつでもプログラムが動作しているという事実が前提で、静的解析はその影を追っているに過ぎません。

PHPStanがどのようなメカニズムで型を付けているのか理解できていないと、自我を失って機械に言われるがままに意図しないコードを書かされる人形になってしまいます。本稿ではPHPStanを自律的に使うために前提となる知識を紹介します。

ここからプログラミング言語と型、そしてPHPについての議論を始めたいのですが、「PHP」という名前はプログラミング言語の名前であり、PHP言語で書かれたソースコードを実行するプログラム名でもあります。ややこしいのですが、単に**PHP**と書くと言語名、**php**と書くとPHPを実行するソフトウェアを指すことにしましょう。

にゃんだーすわん
にゃーん

目次

私とλ

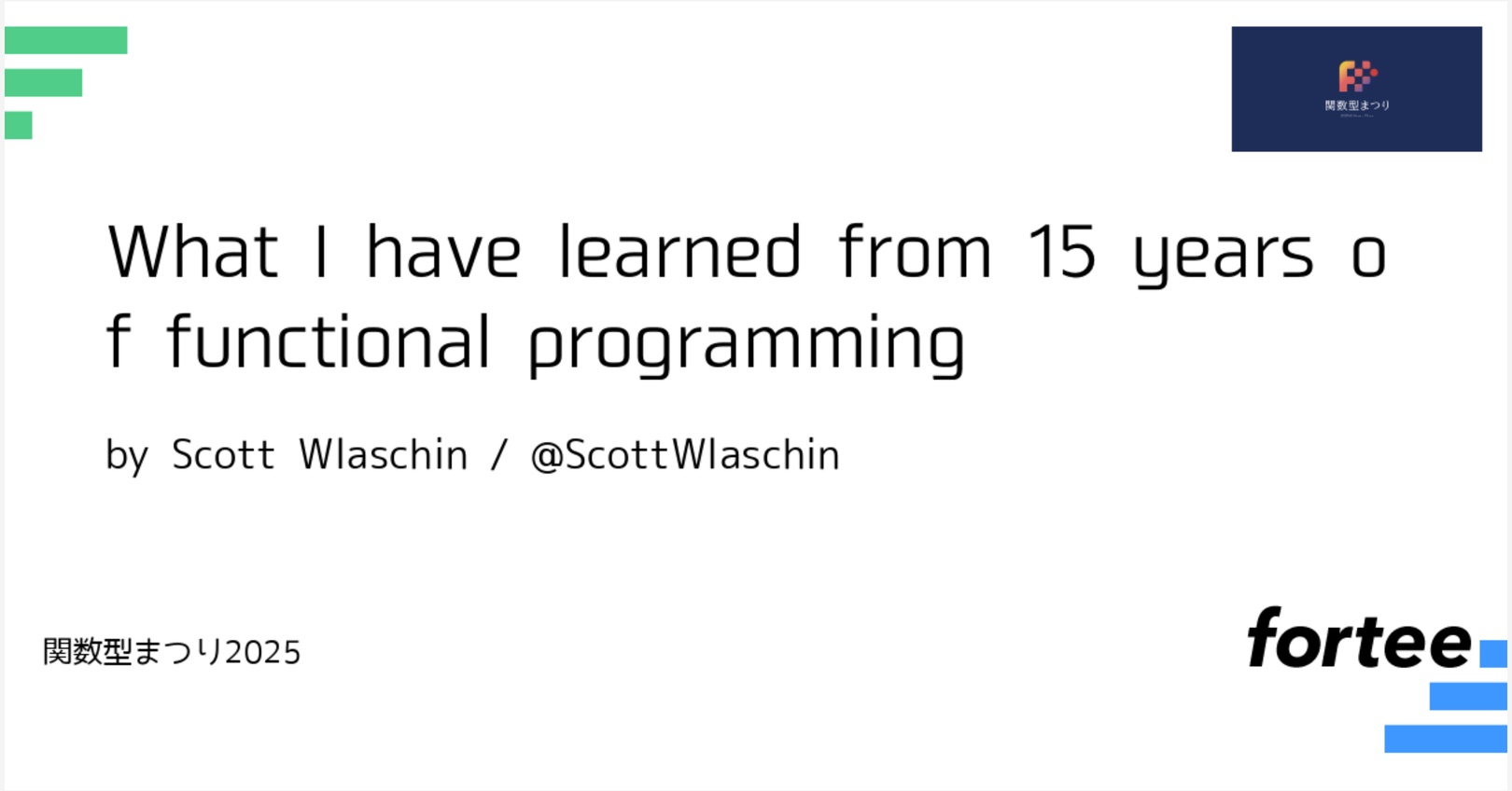
- プログラミング苦手なのでいろんな言語をつまみぐいしてきました
  - 2010年… 魔法言語 リリカル☆Lispに出会う
  - 2011年…「β簡約！λカ娘」(同人誌)を読む
    アルバイト先のF＃マスターに単純型なしラムダ計算を教わる
  - 2012年… SICPを解いてSchemeをちょっとかじる
  - 2013年… Emacs Lispパッケージをごりごり書きはじめる

楽しみましたか？

関数型まつり

2025.6.14 sat – 15 sun

Scott Wlaschinさんの招待講演、最高でしたね…



pixiv

関数型言語
使ってますか？

関数型プログラミング
してますか？

みなさまそれぞれの「関数型」観があると思います

map/reduceが使えたら関数型？

λ関数型言語とはλ (私の偏見に基く認識)

- 関数定義ができれば関数型言語だよ派 ← C言語
- ファーストクラスの関数抽象があれば関数型言語だよ派 ← Python
- 標準ライブラリで高階関数(map/reduce)が扱えればいいよ派 ← Lisp

- 代数的データ型が扱えない言語は関数型言語じゃないよ派　**動的／静的の壁**
- 型推論の完全性がある静的型付き言語が関数型だよ派 ← ML系
- 関数型というのは純粋関数型のことだよ派 ← Haskell
  - モナディックな表示的意味論を持つ言語だよ派

日本人も漢字使ってるし中国語読めるでしょ？

関数型得意だし
Lisp書けるでしょ？

いやいや…
厳しい…

それくらい
「関数型」
アプローチは多様

Lispで「関数プログラミング」してますか？

Lisp族の中ですら
言語(方言)ごとに
文化が違う

みなさんは
Lispを
ご存じですか

歴史 [ソースを編集]

1930年代 [ソースを編集]

関数型言語の開発において、アロンゾ・チャーチが1932年[注釈 1]と1941年[注釈 2]に発表したラムダ計算の研究ほど基本的で重要な影響を与えたものはない[12]。ラムダ計算は、それが考え出された当時はプログラムを実行するようなコンピュータが存在しなかったためにプログラミング言語として見なされなかったにもかかわらず、今では最初の関数型言語とされている[12]。1989年現在の関数型言語は、そのほとんどがラムダ計算に装飾を加えたものとして見なせる[12]。

1960年代 [ソースを編集]

1960年にジョン・マッカーシー等が発表した LISP は関数型言語の歴史において重要である[13]。ラムダ計算は LISP の基礎であると言われるが、マッカーシー自身が1978年[注釈 3]に説明したところによると、匿名関数を表現したいというのが最初にあって、その手段としてマッカーシーはチャーチのラムダ計算を選択したに過ぎない[14]。

歴史的に言えば、 LISP に続いて関数型プログラミングパラダイムへ刺激を与えたのは、1960年代半ばのピーター・ランディン（英語版）の成果である[15]。ランディンの成果はハスケル・カリーとアロンゾ・チャーチに大きな影響を受けていた[15]。ランディンの初期の論文は、ラムダ計算と、機械および高級言語 (ALGOL 60) との関係について議論している[15]。ランディンは、1964年[注釈 4]に、 SECD マシンと呼ばれる抽象的な機械を使って機械的に式を評価する方法を論じ、1965年[注釈 5]に、ラムダ計算で ALGOL 60 の非自明なサブセットを形式化した[15]。1966年[注釈 6]にランディンが発表した ISWIM（If You See What I Mean の略）という言語（群）は、間違いなく、これらの研究の成果であり、構文や意味論において多くの重要なアイデアを含んでいた[15]。 ISWIM は、ランディン本人によれば、「 LISP を、その名前にも表れたリストへのこだわり、手作業のメモリ割り当て、ハードウェアに依存した教育方法、重い括弧、伝統への妥協、から解放しようとする試みとして見ることができる」[15]。関数型言語の歴史において ISWIM は次のような貢献を果たした[16]。

- 構文についての革新[15]
  - 演算子を前置記法で記述するのをやめて中置記法を導入した[15]。
  - let 節と where 節を導入して、さらに、関数を順序なく同時に定義でき、相互再帰も可能なようにした[15]。
  - 宣言などを記述する構文に、インデントに基づいたオフサイドルールを使用した[15]。
- 意味論についての革新[16]
  - 非常に小さいが表現力があるコア言語を使って、構文的に豊かな言語を定義するという戦略を導入した[15]。
  - 等式推論 (equational reasoning) を重視した[15]。
  - 関数によるプログラムを実行するための単純な抽象機械としての SECD マシンを導入した[16]。

出典: フリー百科事典『ウィキペディア(Wikipedia)』関数型プログラミング より引用

1958年登場の言語
(現役では最古級)

C言語

```
puts("foo")
```

Ruby

```
puts "foo"
```

Lisp

```
(puts "foo")
```

(imaginary)

(((((LISP)))))

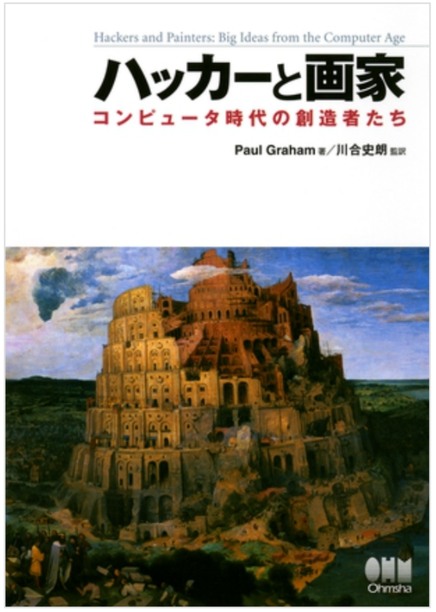
(((カッコが多い)))

カッコだらけの構文 (S式)

- （a b c）のように並べたリストを構文の基本単位とする
  - 式と文の区別はない。全てが式。
- このリストによるコードをS-expressionと呼ぶ
  - 日本語では**S式**（またはS表現とも）
- Lisp族の言語の構文に基礎であるほか、
  WebAssembly テキスト形式（.wat）でも採用されている

結構ひとによって
印象が違う
言語だと思う

ハッカーと画家 コンピュータ時代の創造者たち



著者　：Paul Graham 著、川合 史朗 監訳
定価　：2,640円 （本体2,400円＋税）
判型　：A5
頁　：280頁
ISBN　：978-4-274-06597-2
発売日　：2005/01/26
発行元　：オーム社

普通のやつらの上を行け ---Beating the Averages---

著者：Paul Graham

これは、Paul Graham: Beating the Averages を、原著者の許可を得て翻訳・公開するものです。

プロジェクト杉田玄白正式参加テキスト。

<版権表示>
本和訳テキストの複製、変更、再配布は、この版権表示を残す限り、自由に行って結構です。
(「この版権表示」には上の文も含まれます。すなわち、再配布を禁止してはいけません)。

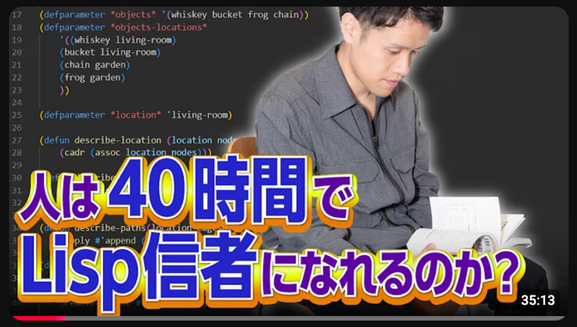
原文: http://www.paulgraham.com/avg.html
日本語訳：Shiro Kawai (shiro @ acm.org)
<版権表示終り>

ゆるコンピュータ科学ラジオ Lisp

作成

すべて | ショート | 動画 | 未視聴 | 視聴済み | 最近アップロードされた動画 | ライブ

フィルタ



ゆるコンピュータ科学ラジオ ポール・グレアム

作成

すべて | ショート | 動画 | 未視聴 | 視聴済み | 最近アップロードされた動画 | ライブ

フィルタ

プログラミング言語への愛があれば、未来予知できるし50億稼げる【ポール・グレアム2】#111

8.3万 回視聴・1 年前

ゆるコンピュータ科学ラジオ

「ポール・グレアム」の第2回です。「iPhoneとSaaSの到来を予想した男」「普通の環境に抗った奇妙な言語の狂信者」「愛が ...

チャプター数: 9 便利すぎるツールを作った人に感謝したい | 現代の情報技術を正確に予測したい | 意識高い界隈...

伝説のIT起業家からボロ儲けの技術を学ぶ。〇〇すれば50億稼げるらしい【ポール・グレアム1】#110

9.1万 回視聴・1 年前

ゆるコンピュータ科学ラジオ

新シリーズ「ポール・グレアム」です。「スタートアップ界隈で超有名な起業家」「成功に導くありえない働き方」「ハッカーと ...

チャプター数: 6 将来、お金持ちになるには？ | コンピュータサイエンスで大富豪になれる？ | 知名度皆無のポー...

Lispはなぜ狂信者を生むのか？ その答えは、奇妙な出自。【ポール・グレアム3】#112

9万 回視聴・1 年前

ゆるコンピュータ科学ラジオ

「ポール・グレアム」の第3回です。「Lispはなぜ狂信者を生むのか」「文法よりも意味を重視した言語」「最後までチョケ ...

チャプター数: 7 自分のうんざりベンチマークとは？ | ポール・グレアムはスーツ嫌い | Lisperたちが狂信するLisp...

Lispはなぜ神の言語と呼ばれるのか

古から存在するプログラミング言語Lisp。その通り名の一つに「神の言語」というものがある。

なんという甘美な響きだろうか。この名前だけで男子中学生は全員習得することを決心するだろう。そのくらいインパクトのある名前である。文部科学省はプログラミング教育を推進するにあたって、この通り名を公式に採用することを真剣に検討されると良いと思う。

さて、「神の言語」の勉強を始めてみて驚くだろう。どのあたりが神なのかよくわからないのだ。たしかに丸括弧は多い。多いが、それと神との関連性はよくわからない。

何か土着の信仰の話なのだろうか？あなたがそう戸惑うのも無理からぬことだ。

私自身、前からこのフレーズは聞いたことがあったが、あまり理解していなかった。そこで今回改めて調べてみたので共有しようと思う。

「神の言語」の由来

そもそも、Lispは神の言語だと誰が言い出したのか？どうやら海外でそのような歌が作られたというのが始まりのようである。Bob Kanefskyという方が”Eternal Flame”というタイトルで、2000年にリリースされたそうである。Lispの歌だって！？いきなりの濃いエピソードに、ただただ困惑するばかりである。Pythonの歌とか、C++の歌とか聞いたことがあるだろうか？[1]

あまりに狂信的なエピソードなので、やっぱり土着の信仰なのかもしれないと少し心配になってくるが、まずは中身を見てみるとしよう。

日本語訳を掲載されているページがあったので紹介する。

Twitter

自作記事一覧

Julia言語で入門するプログラミング

Juliaの素晴らしきエコシステム

Lispはなぜ神の言語と呼ばれるのか

怖いくらいにわかりやすい文字コードの解説

ユニコードの全角と半角を判定しよう

JuliaとLispのマクロの比較

Juliaのマクロは凄かった。Lispは果たして勝てるのか？

【写真】天才プログラマに聞く10の質問（1）

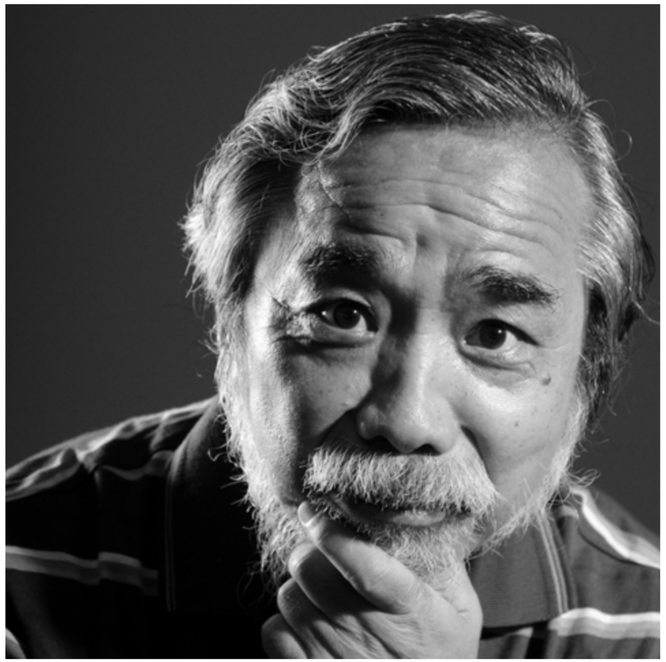
Lispの仏さま 竹内郁雄の目力

「天才」。世の中にはそう呼ばれている人たちがいる。本連載では、これまで数々の偉業を成し遂げてきた天才プログラマに、スキルやキャリアに関する10の質問をする。彼／彼女らのプログラマとしての考え方・生き方とは。天才の言動から見えることとは何か。

2008年07月23日 00時00分 公開

[荒井亜子，@IT]

第1回の天才プログラマは、いわずと知れたLispの大御所ハッカー、竹内郁雄氏。



竹内郁雄氏（61歳）　撮影：大星直輝

竹内氏によると、Lispはすべての言語の原点なのだという。「いまRubyがブームだが、Rubyは要するにカッコのないLisp。XMLもぶ厚いカッコのあるLisp。いろいろなプログラミング技法を見ますが、『Lispにあった』ということが多いです」（竹内氏）

（Lispって結局何-p）

虚実に包まれたLisp

pixiv

ということで

関数型まつり2025

採択 2025/06/15 17:30〜 Track B 公募セッション10分（LT） Beginner 周辺ツール 言語処理系 入門

Lispは関数型言語(ではない)

うさみけんた tadsan

☆ 4

対象とする聴衆のレベル（該当するレベルを記載してください。）

- Beginner: 分野の前提知識を必要としない

セッションのテーマ（該当するテーマを記載してください。なければ追加頂いて良いです）

- 入門
- 周辺ツール（ビルドツール、静的解析ツール、エディターなど）
- 言語処理系（コンパイラー、インタープリターなど）

セッションの概要

LISPは記号処理に特化して初期の人工知能研究を牽引した言語であり、実用的な関数プログラミングを提供した最初の言語といえるでしょう。
現在のLisp族の末裔たちも関数プログラミングのための機能を備えている一方、後続の関数型言語(ML族)が持つ静的型や参照透過性、モナドなどの機能は持っていません。

このトークでは日常的にEmacsとEmacs Lispを利用しているユーザーが、Lispのどのような側面が「関数型」的で、なぜ関数型言語ではないと考えているのか、Emacsを用いたプログラミングの実践とEmacs Lispの「正体」までお話しします。

関数型言語
(じゃないの！？？)

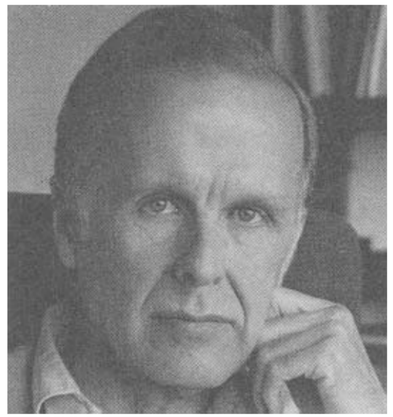
話をきいてください

そもそも
関数型って
なんだ

「関数型」と呼んだのは ジョンバッカス (1977年)

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

"Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs: Communications of the ACM: Vol 21, No 8"より引用

combinatory logic

コンビネータ論理

(1929年-)

pixiv

λ-calculus

ラムダ計算

(1930年代-)

貳佰伍拾陸夜日記

ni. (031) hyaku (71) go (797) jyu (226) roku (565) ya. (092) ni. (607) ki. (101)

2010-02-08

ラムダ計算基礎文法最速マスター

article lambda lang

ラムダ計算は, 多くのプログラミング言語, とくに関数型言語の原形になっています. ラムダ計算について理解しておくことは, 多くのプログラミング言語の習得に役立つでしょう.

ラムダ計算はチューリング完全で, 計算能力としてはふつうのプログラミング言語と同じです. ラムダ計算で計算を書く訓練をしておくことは, 任意の計算を関数のみを使って(他の制御構文を用いずに)書くときに役立ちます. ふつうに書いたら煩雑な処理を, 関数型言語のやり方で書くとすっきりすることが多々あり, コードを自由自在に書くためには必須の考え方と言えるでしょう.

項

ラムダ計算の式を**項(term)**と言います. 項は変数, 抽象, 適用のいずれかです.

変数

変数(variable)はふつう1文字で書きます. 変数には関数内の**束縛変数(bound variable)**か**自由変数(free variable)**かという区別があります. 関数を値に適用する以外のやり方で変数に値を代入することはできません.

```
x      # 変数は項
λx.x   # xは束縛変数
λx.y   # yは自由変数
```

プロフィール

id:tarao
ソフトウェアエンジニア

読者です 157

@oaratをフォロー

（λ計算）

$$\lambda x.\lambda y.x$$

(Haskell)

$$\lambda x.\lambda y.x$$

```
\x -> \y => \x
```

(const)

(JavaScript)

$$\lambda x.\lambda y.x$$

```
x => y => x
```

λx.λy.x

(太古のJavaScript)

```
function (x) {
    return function (y) {
        return x;
    };
};
```

(PHP)

$$\lambda x.\lambda y.x$$

```
fn($x) => fn($y) => $x
```

単純型なしλ計算は
チューリング完全で
あることが知られている

関数と適用しかないが
工夫して制御構造も
データ構造も作れる

こういう積み重ねで現代の関数型っぽい要素が整理された

レキシカルスコープとか
不変性とかカリー化とか
型推論とか代数的データ型
とか遅延評価とかモナドとか
関数合成とか参照透過とか

で、Lispはどういう関数型言語なの？

λ関数型言語とはλ (私の偏見に基く認識)

- 関数定義ができれば関数型言語だよ派 ← C言語
- ファーストクラスの関数抽象があれば関数型言語だよ派 ← Python
- 標準ライブラリで高階関数(map/reduce)が扱えればいいよ派 ← Lisp

- 代数的データ型が扱えない言語は関数型言語じゃないよ派　**動的／静的の壁**
- 型推論の完全性がある静的型付き言語が関数型だよ派 ← ML系
- 関数型というのは純粋関数型のことだよ派
  - モナディックな表示的意味論を持つ言語だよ派 ← Haskell

関数型的な主観だけで比較してしまうとLispとPHPは同等

Lispは関数型言語ではない

Lisp is **NOT** a functional language

pixiv Inc.
USAMI Kenta

pixiv

2025-06-15 #fp_matsuri
関数型まつり 2025

λ-calculus

ラムダ計算

(1930年代-)

pixiv

LISPは
ラムダ計算の
実装ではない

(すくなくとも初期は)

CHAPTER | FREE ACCESS

History of LISP

Author: John McCarthy | Authors Info & Claims

History of programming languages • June 1978 • Pages 173 - 185 • https://doi.org/10.1145/800025.1198360

Published: 01 June 1978 Publication History

IV

LISP Session

Chairman: Barbara Liskov
Speaker: John McCarthy
Discussant: Paul Abrahams

LISP ≠ LAMBDA CALCULUS

How do you condense a 30 minute talk in 5 minutes? Should you even try? These are the questions I struggled with when someone nudged me to register for the lightning talks. My talk was 30 minute long because I was to jump in if a last-minute incident would prevent someone to get on stage. Call me the backup speaker, if you will. In organizing *Heart of Clojure*, Arne Brasseur and Martin Klepsch had prepared for every eventuality. Luckily, the event was incident free, and I was relieved of duty. But that someone was right: a lightning talk was the only redeeming option I had before calling it quits.

TL;DR Lisp is not a realization of the Lambda Calculus

derivative of the *Lambda Calculus*, I burst onto the stage asking for a show of hands: *Who thought Lisp was based on the Lambda Calculus? Who thought it wasn't? Who didn't know what to think?*

> *… one of the myths concerning LISP that people think up or invent for themselves becomes apparent, and that is that LISP is somehow a realization of the lambda calculus, or that was the intention. The truth is that I didn't understand the lambda calculus, really. — John McCarthy, Lisp session, History of Programming Languages*

Note: Yes, John McCarthy borrowed the lambda notation from Alonzo Church. He also understood it better than he wants you to believe.

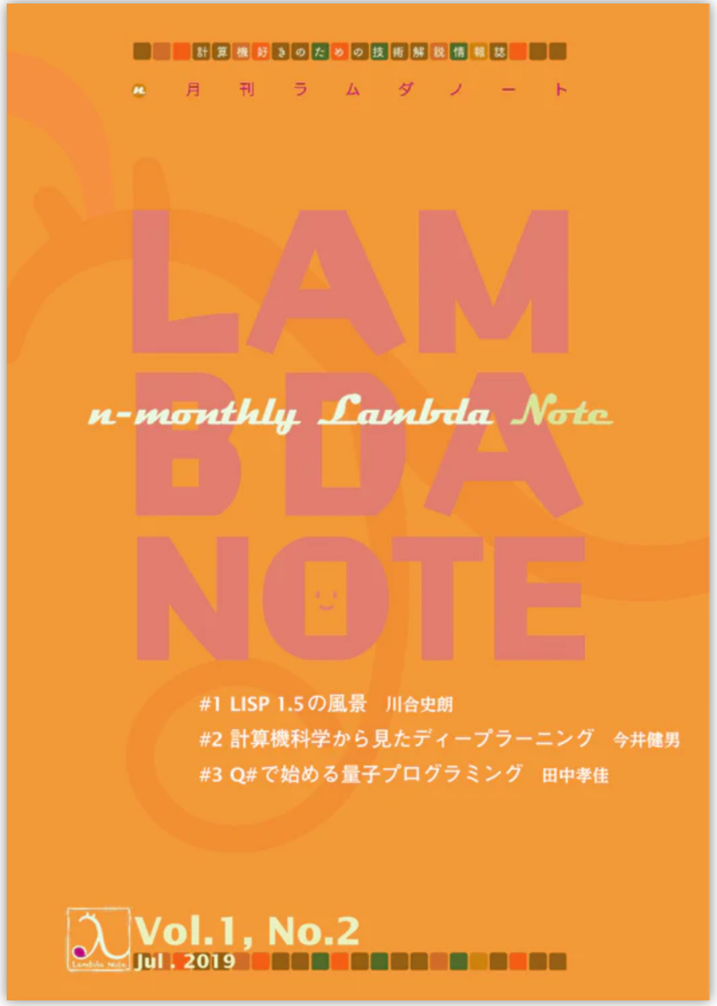
Lambda Calculusの派生語であるという誤解が広まっている
ことに賭け、ステージに飛び出して挙手を求めました。Lisp
が *Lambda Calculus に基づいていると思う人はいますか？
そうでないと思う人はいますか？どう考えたらいいのかわ
からない人はいますか？*

> *...LISPに関する、人々が勝手に考え出したり、作り出したり
> している誤解の一つが明らかになりました。それは、LISPは
> ラムダ計算の実現形である、あるいはそれが意図されていた
> というものです。実のところ、私はラムダ計算を全く理解し
> ていませんでした。– ジョン・マッカーシー、Lispセッショ
> ン、プログラミング言語の歴史*

注：そうです、ジョン・マッカーシーはラムダ記法をアロ
ンゾ・チャーチから借用しました。そして、彼は自分が思
っている以上にラムダ記法を理解していました。

LISPと変数スコープ

- 初期のLISP(LISP1～MACLISP)は変数スコープの概念が未確立
  - レキシカルスコープかダイナミックスコープかが定かではない(!)
    FUNARG問題
- 1975年に登場したSchemeではクロージャ(レキシカルスコープ)が導入
- 1984年に登場したCommon Lispはレキシカルスコープが基本だが、
  スペシャル変数として定義することでダイナミックスコープも使える








n月刊ラムダノート Vol.1, No.2(2019)

¥1,500（税込¥1,650、送料当社負担）

カートに追加

- 紙書籍をお届けします（PDFがついてきます）
  - PDFのみ必要な場合は、こちらからPDF単体をご購入ください
- 通常はご注文から2〜3営業日で発送します。
- 年末年始や大型連休など、1週間から10日程度、配送のお休みをいただく場合があります。詳しくはお知らせをご覧ください。

計算機好きのための技術解説情報誌

- エヌゲッカンラムダノート（不定期刊行）
- 78ページ
- A5判
- 紙書籍は1色刷
- 2019年7月8日 第1巻第2号／通巻2号 発行

n月刊ラムダノートは、*n*ヶ月ごとに刊行される、計算機好きのための技術解説情報誌。コンセプトは「いろんなIT系技術書から1章ずつ選んできた解説記事の集まり」です。毎号3つから4つの記事をお届けします。

目次

#1 LISP 1.5の風景（川合史朗）

プログラミング言語について勉強していくと、いつか必ず出会うLisp。John McCarthy が発明したLisp には、Scheme やCommon Lisp をはじめとする多様な方言があり、現在でも多くの実用的な処理系が世界中で利用されている。

それらLisp 処理系の源流のひとつは、最初期のLisp 実装にさまざまな改良を施したLISP 1.5 である。本稿では、Scheme の処理系Gauche の作者である川合氏により、Gauche によるLISP 1.5 の実装を通じて“LISP 1.5 Programmer’s Manual” の世界を案内していただく。インタプリタの実装例としてのみならず、日ごろLisp 族の言語を利用している人でも現在はあまり意識することがないM式やFUNARG 問題といった初期のLisp をめぐる話題にも触れる。（編集部）

(PHP)

$$\lambda x.\lambda y.x$$

```
fn($x) => fn($y) => $x
```

(Scheme)

$$\lambda x.\lambda y.x$$

```
(lambda (x)
  (lambda (y)
    x))
```

(Racket)

$$\lambda x.\lambda y.x$$

```
(λ (x) (λ (y) x))
```

Lispでも
ラムダ計算できるが
書きやすくはない

関数の純粋性を担保する仕組みが不足している

とはいえ
関数プログラミングの
道具は揃っている

pixiv

その道具を
どう使うかによって
Lispの文化がある

LISPとLisp系言語

- 「方言」と呼ばれるが、C言語とC#とJavaとJavaScriptくらい違う
  - Scheme:　比較的シンプルな言語仕様と衛生的マクロ
  - Common Lisp:　機能豊富で処理系も多い
  - Clojure:　関数型重視でJVMで動く。Rich Hickeyの哲学
  - Emacs Lisp:　GNU Emacsで動く。Emacsそのもの

```clojure
(defn sum-to-n [n]
  "1からnまでの整数の総和を計算する."
  (reduce + (range 1 (inc n))))
```

U:**- 2025-06-15-143920-sum.clj (4,1) [1] (Clojure

```scheme
(define (sum-to-n n)
  "1からnまでの整数の総和を計算する."
  (let loop ((i n) (acc 0))
    (if (zero? i)
        acc
        (loop (- i 1) (+ acc i)))))
```

U:**- 2025-06-15-144417-sum.scm (2,37) [1] (Scheme G

```lisp
(defun sum-to-n (n)
  "1からnまでの整数の総和を計算する."
  (loop for i from 1 to n
        sum i))
```

U:--- 2025-06-15-144121-sum.l (2,35) [1] (Lisp Pr

```elisp
(defun sum-to-n (n)
  "1からnまでの整数の総和を計算する."
  (let ((sum 0)
        (i 1))
    (while (<= i n)
      (setq sum (+ sum i))
      (setq i (1+ i)))
    sum))
```

-:**- 2025-06-15-144224-sum.el (8,9) [1] (ELisp/d

pixiv

機能的には
どのLispでも
他のスタイルで書ける

つくって学ぶプログラミング言語 Rubyによる Scheme処理系の実装

渡辺昌寛

達人出版会
517円 (470円+税) PDF EPUB

プログラミングをより深く理解するための近道は、プログラミング言語を実装してみること。Schemeのサブセットを Rubyで実装していくことで、プログラムはどう実行されるのか、その基本がはっきり分かります。

概要 サンプル リンク用タグ

ページサンプル閲覧

以下はPDF版の先頭ページなどから、一部を抜粋して縮小した画像です。
（※EPUB版を提供している書籍の場合、基本的に内容は同じですが、見映えは異なります。 またお使いのEPUBリーダによっても見え方は大きく異なることがあります。）

tallesl / Rich-Hickey-fanclub

Type / to search

Code Issues Pull requests Security Insights

Rich-Hickey-fanclub

Public

README

A collection about Rich Hickey's works on the internet.

Work

- Clojure
- Datomic
- Harmonikit

He also made other not as well known lisps prior to Clojure.

トップ > clojure > Clojureと「Simple Made Easy」

＊ ＊ ＊

2017-05-15

Clojureと「Simple Made Easy」

clojure

プログラミング言語というのは、その作者が理想とする世界に合うようにデザインされているものだから（みんな信じないかもだけど、Javaですらそうなのですよ）、Clojureのことを理解するには、作者であるRich Hickeyのプログラミング観を知るのが手っ取り早いでしょう。

Rich Hickeyはさまざまなプレゼンテーションを発表していて、多くはネットで見られます。示唆に富んで皮肉も効いてておもしろいので、ファンも多くて、彼の独特の髪型（往年のロック歌手風）からか、「Rich Hickey's Greatest Hits」というブログ記事もあったりします（プレゼンテーション動画へのリンク集です）。

初めての人のためのLISP［増補改訂版］

1点すべての画像を見る

竹内 郁雄 著

形式：**書籍**
発売日：2010年03月09日
ISBN：9784798119410
定価：3,278円（本体2,980円＋税10%）
仕様：A5・384ページ
カテゴリ：プログラミング・開発
キーワード：#プログラミング,#開発環境,#開発手法,#Web・アプリ開発

紙の書籍 電子書籍

立ち読み

Lisp の自己拡張性のもう一つの意味は，その抽象化能力の高さである。新しい関数やマクロの定義はそのまま言語の抽象語彙や構造を増やす。言語の骨格を変えないで，概念の抽象化がどんどん進む。抽象化能力を指しているものではないかもしれないが，Lisp は泥の玉，だからそれにいくら泥をくっつけてもやはり泥の玉という「誉め言葉」が昔あった。雑草のような生命力をもった言語ともよく言われる。この能力は Lisp にオブジェクト指向が入ることによってさらに増進された。残念ながら，Lisp に導入されたオブジェクト指向について本書では触れない。

Fortran や Cobol が生まれたての 1950 年代末，仲間たちと Lisp を練り上げる前の構想段階では，McCarthy は関数自体を操作できる（今日的な意味での）関数型の言語であると同時に，論理指向の言語であるべきと考えていた。後者は形にならなかったが，Lisp が世界で初めての関数型プログラミング言語であったことは間違いない。そのことをもって，いまでも Lisp を関数型言語に分類する向きがあるが，関数型言語の特徴を備えた手続き型言語というほうが実態に合っている。教養の言語として Lisp を教えるときに，純粋関数型言語の特徴だけをゴリゴリ押しまくるのはよい方針ではない。Lisp の雑草のような生命力の根源を見失わせる恐れがあるからだ。本書を書くにあたって，そのあたりのことが伝わるように工夫したつもりである。

「初めての人のためのLISP［増補改訂版］」より引用
竹内郁雄著，2010年3月9日 初版第1刷発行，翔泳社刊
（電子書籍版を底本とする）

Lispは関数型言語
ではないが
能力と文化によって
実現している

pixiv

Emacs Lisp

広く普及したLispでありながら
Lispらしくないと敬遠された

RESEARCH-ARTICLE | OPEN ACCESS

Evolution of Emacs Lisp

Authors: Stefan Monnier, Michael Sperber | Authors Info & Claims

Proceedings of the ACM on Programming Languages, Volume 4, Issue HOPL • Article No.: 74, Pages 1 - 55
https://doi.org/10.1145/3386324

Published: 12 June 2020 Publication History

Abstract

While Emacs proponents largely agree that it is the world's greatest text editor, it is almost as much a Lisp machine disguised as an editor. Indeed, one of its chief appeals is that it is *programmable* via its own programming language. Emacs Lisp is a Lisp in the classic tradition. In this article, we present the history of this language over its more than 30 years of evolution. Its core has remained remarkably stable since its inception in 1985, in large part to preserve compatibility with the many third-party packages providing a multitude of extensions. Still, Emacs Lisp has evolved and continues to do so.

Important aspects of Emacs Lisp have been shaped by concrete requirements of the editor it supports as well as implementation constraints. These requirements led to the choice of a Lisp dialect as Emacs's language in the first place, specifically its simplicity and dynamic nature: Loading additional Emacs packages or changing the ones in place occurs frequently, and having to restart the editor in order to re-compile or re-link the code would be unacceptable. Fulfilling this requirement in a more static language would have been difficult at best.

Emacs Timeline.

by Jamie Zawinski <jwz@jwz.org>
written: 8-Mar-1999
updated: 29-Oct-2007

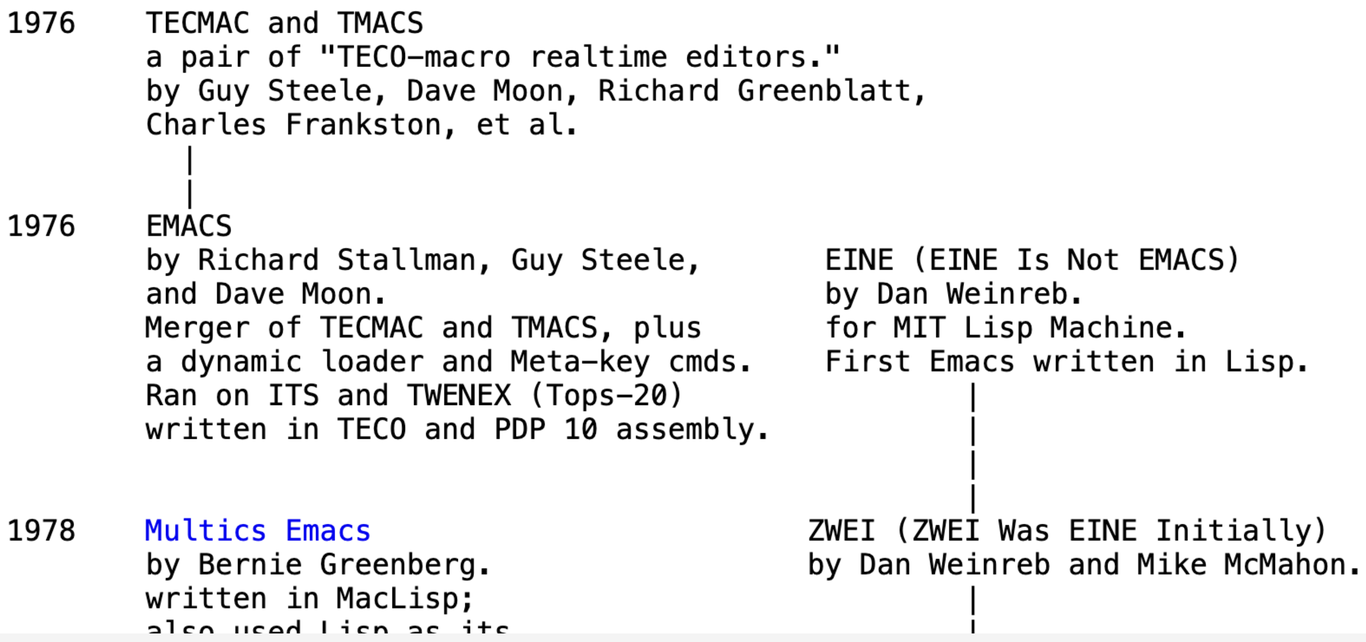
Someone prodded me into drawing up a timeline of the Emacs family tree. Let me know if you have any additions/corrections.

This isn't intended to be a complete list of everything that has ever called itself Emacs -- as Craig Finseth's periodic posting of emacs implementations shows, that would be much larger than this.

However, I think this is fairly accurate for the GNU/Lucid/X Emacs part of the world, and their important predecessors. (An explanation of how and why the FSF Emacs / Lucid Emacs split came about is over here.)

For more detail about the early days, please see Bernie Greenberg's paper, Multics Emacs: The History, Design and Implementation.

I've drawn lines only where code is shared, not merely ideas.



Emacs Lispの弱点

- クロージャと真正のレキシカルスコープがなかった
  - マクロで擬似的に再現したレキシカルスコープはあった
- 末尾再帰最適化がない
- カジュアルに副作用がある
- 名前空間（パッケージ）がない
- リーダーマクロがない
- Emacsと密結合している

Emacs Lisp (elisp)

- Richard Stallman (RMS)のこだわりのLisp方言
  - RMSが最近も機能追加しようとしている
- 現存するメジャーな方言ではCommon Lisp寄りではあるが結構違う
  - Lisp Machine LispやGosling MLispの影響がある
  - Common Lisp風の機能はcl-libという標準ライブラリで同梱

Emacs

- Lispを内蔵したエディタというべきか、
  言語処理系にエディタ機能がついているべきか
- init.el というファイルにLispで設定を書くが、
  初期化設定とEmacsで動かす機能の開発には境目がまったくない
- Lispが動く仮想的なLispマシンそのもの
  - Evilという互換性が極めて高いVimインターフェイスが実装されている

Emacs Lispの弱点

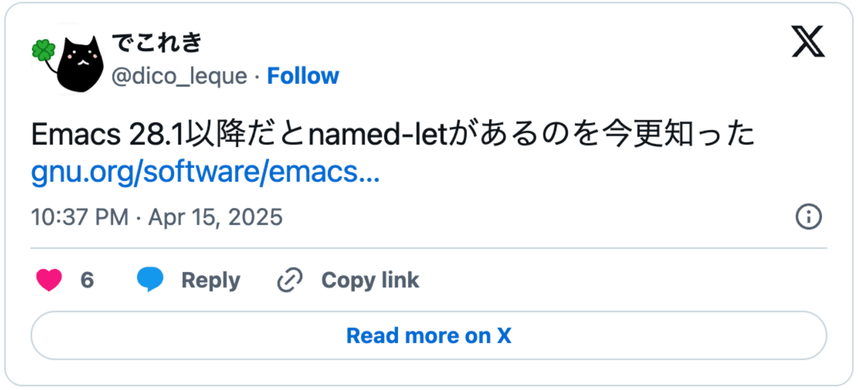
- ~~クロージャと真正のレキシカルスコープがなかった~~
  - ~~マクロで擬似的に再現したレキシカルスコープはあった~~
- ~~末尾再帰最適化がない~~
- ~~カジュアルに副作用がある~~
- 名前空間（パッケージ）がない
- リーダーマクロがない
- Emacsと密結合している

Emacs Lispにnamed-letが追加されてた

2025/06/12に公開

Emacs scheme lisp Tech

つい最近まで知りませんでした...



Schemeを使っている方には「`let`で末尾再帰するときのアレをEmacsでも使えるようにする」というので通じるとは思うのですが...

にゃんだーすわん
にゃーん

目次

```
(defsubst looking-at-p (regexp)
  "\
Same as `looking-at' except this function does not change the match data."
  (declare (side-effect-free t))
  (looking-at regexp t))

(defsubst string-match-p (regexp string &optional start)
  "\
Same as `string-match' except this function does not change the match data."
  (declare (side-effect-free t))
  (string-match regexp string start t))
```

- Emacsの検索には副作用があるが -p のついた述語関数で避けられる
- (declare (side-effect-free t)) で純粋関数としてマークできる

Emacs Lisp

かつては不出来な末っ子扱い
現在は必死に追い上げている

Lispは
マルチパラダイム

Lisp Machine

AIブームを牽引した専用機
(1970年代後半-)

そういう機械が
40年以上前にあった

図1 AI ワークステーション ELIS

現在，ELIS は NTT，沖電気工業などが出資して作った NTT の子会社 NTT インテリジェントテクノロジ (NTT-IT) から約1000万円で販売されている．(この価格は Lisp マシンとしてはかなり安いと思うが，EWS として考えると高い．今後こういうマシンの真の競争相手は既存の Lisp マシンではなく UNIX ベースの EWS になるはずだから，マルチユーザで AI 向きという付加価値を加味してももう少し安いことが望まれる．もっとも，私は NTT-IT の経営に口をはさむ立場ではないので，これはあくまでも私の個人的願望だ．) というわけで，商品としての TAO/ELIS と我々 (主に TAO 周辺のソフトウェア担当者) の研究材料としての TAO/ELIS の切り分けが一時ちょっと微妙だったが，現在いろいろな関係が整理され，我々はまた研究材料としての TAO/ELIS に没頭できる状態になっている．今後の我々の研究の成果の中にもし「使える」ものがあれば，それを世に出すことのできるパイプが布設されたというわけである．

NTT ソフトウェア研究所の私の同僚たちは，これから TAO/ELIS の上に知能処理のための統合的プログラミング環境 NUE (New Unified Environment) を作り上げる研究を進めようとしている．NUE のキャッチフレーズは，「知能処理ソフトウェアの研究開発の場において研究者や開発技術者の創造性を最大限に発揮させる高性能・高機能のプログラミング環境」となかなか勇ましくて格好いいのだが，いまところ具体的なイメージは定まっていない．NUE (鵺) という名前からしてこれは当然かもしれない．ある人が NUE は NUE's Undefined Environment (cf. GNU's Not Unix) のことではないかと鋭く指摘してくれたが正鵠を突いている．

「Multi-Paradigm Language TAO」より引用
https://www.nue.org/nue/tao/bitao/s001.html

この写真は国産LISPマシンのELISなので本家Emacsの開発経緯とは直接関係ないのだが内蔵されたZENというEmacs風独自エディタの日本語入力メソッドKanzenが後のSKKに繋がりIME史に影響を及ぼしている

マルチパラダイム言語 TAO

竹内 郁雄

pixiv

【写真】天才プログラマに聞く10の質問（1）

Lispの仏さま 竹内郁雄の目力

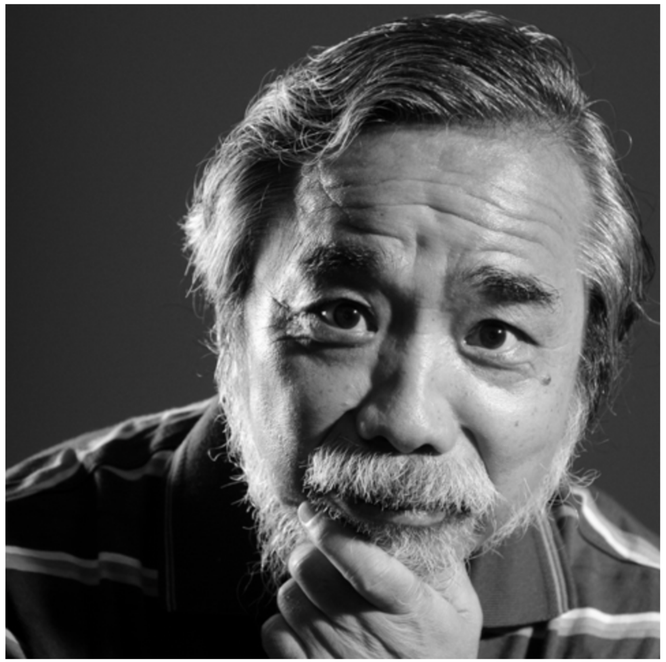
（1/2 ページ）

「天才」。世の中にはそう呼ばれている人たちがいる。本連載では、これまで数々の偉業を成し遂げてきた天才プログラマに、スキルやキャリアに関する10の質問をする。彼／彼女らのプログラマとしての考え方・生き方とは。天才の言動から見えることとは何か。

2008年07月23日 00時00分 公開

[荒井亜子，@IT]

第1回の天才プログラマは、いわずと知れたLispの大御所ハッカー、竹内郁雄氏。



竹内郁雄氏（61歳）　撮影：大星直輝

竹内氏によると、Lispはすべての言語の原点なのだという。「いまRubyがブームだが、Rubyは要するにカッコのないLisp。XMLもぶ厚いカッコのあるLisp。いろいろなプログラミング技法を見ますが、『Lispにあった』ということが多いです」（竹内氏）

3.2 メッセージ伝達

前の節でも述べたことから推察できるように，TAO のメッセージ伝達式は

```
(レシーバ セレクタ 引数 ...)
```

という形をしている．この式は，**レシーバ**となるオブジェクトがセレクタ以降のメッセージ (セレクタと引数の並びを包括したものを**メッセージ**と呼ぶ) を受け取ることを表わす．ここで，レシーバと引数はすべて評価されるが，セレクタは評価されない．

```
(3 '+ 5)
```

と書きたくないからである．

外見上，メッセージ伝達式と Lisp のふつうの関数呼び出しには区別がない．たとえば，

```
(fn move x y)
```

と書かれると，これがメッセージ伝達なのか，関数呼び出しなのか見ただけでまず，インタプリタが

```
(α β γ ...)
```

という式をどう解釈実行するかを**図3**に「フローチャート」で示す．Lisp の元祖 John McCarthy 大先生にこのフローチャートを見せたら，「Lisp の意味を説明するのにフローチャートを使うのは見たことがない」と言って随分機嫌を害したようだったが，どういうわけかこの説明にかぎっては私は昔からフローチャートを使うのが好きだ．さて，ここで重要なのは，αが関数シンボルでなければαを1回評価することである．

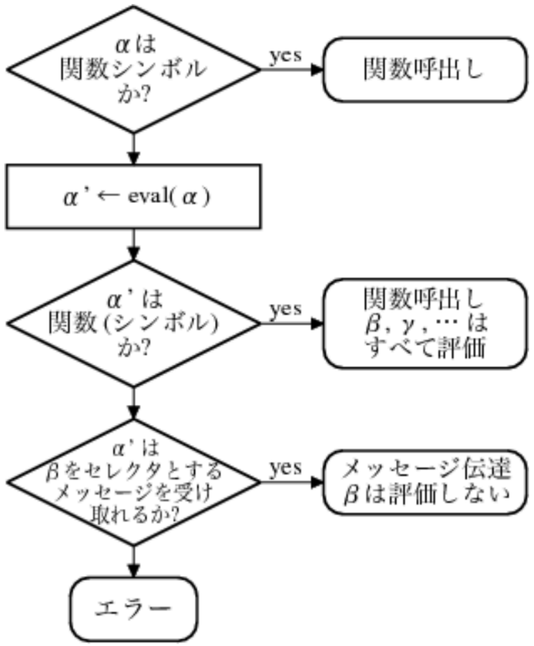


図3 式 (α β γ …) の評価

これはインタプリタだからこそ許されるが，コンパイラにとっては非常に困るものである．Common Lisp はもちろんこんなことを許さない．αを評価した結果がメッセージを受け取ることのできるオブジェクトであれば，この式はメッセージ伝達と解釈される．もう1つ注意してほしいのは，メッセージ伝達かどうかの判断がふつうの Lisp の関数呼び出しの解釈実行になんら影響または負荷を及ぼしていないことである．それは当たり前で，メッセージ伝達に至る道筋は従来の Lisp でエラーになっていたところにほかならない．

とはいうものの，一見しただけで関数呼び出しかメッセージ伝達かがわからないのではコンパイラが途方に暮れる．TAO のコンパイラはこういう場合，ほかをすべてコンパイルし，わからない式だけを S 式のまま残しインタプリタで解釈実行するようなハイブリッドコードを出す．ハイブリッドコードでもちゃんと実行できるところが TAO の面白いところだが，コンパイルしたのにメッセージ伝達の速度が上がらないという珍現象が起こってしまう．Flavors と同じように send 関数を使ってメッセージ伝達をやればこの問題は起こらないが，元の木阿弥である．そこで TAO にはもう1つのメッセージ伝達式の形式を用意した．それは次のように丸カッコではなく，角カッコを使う．

```
[レシーバ セレクタ 引数 ...]
```

TAO/ELISの特徴は現代Lispには受け継がれていない

どのLispも多かれ
少なかれ
マルチパラダイム

Lispは
関数型の枠に
留まっていない

Lispのパワーの根源は S式・マクロ

pixiv

emacs-elsa / Elsa

Type / to search

Code | Issues 89 | Pull requests 6 | Discussions | Actions | Projects | Security | Insights

Elsa Public

Sponsor | Unwatch 20

README | GPL-3.0 license

Elsa - Emacs Lisp Static Analyser

test failing

(Your favourite princess now in Emacs!)

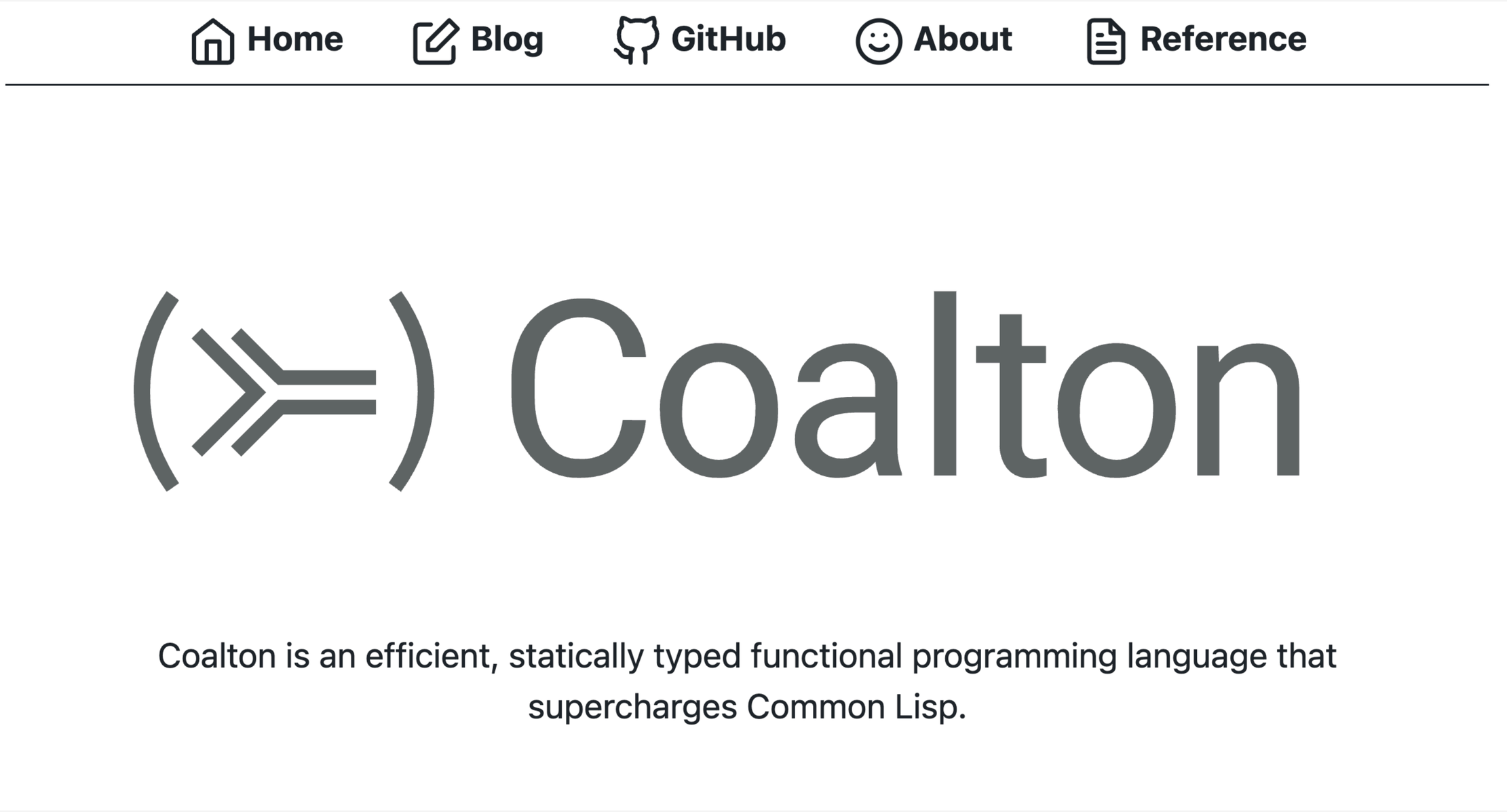
coverage 65% | PayPal Donate | Patreon Become a patron

Elsa is a tool that analyses your code without loading or running it. It is 100% side-effect free and we strive to keep it that way, so you can analyse any elisp code from anywhere safely.

Elsa adds a powerful type system on top of Emacs lisp (completely optional). In can track types and provide helpful hints when things don't match up before you even try to run the code.

Table of Contents

- Motivation


Home
Blog
GitHub
About
Reference
(≫=) Coalton
Coalton is an efficient, statically typed functional programming language that supercharges Common Lisp.

May 26, 2025 blog

Hacker News now runs on top of Common Lisp

Hacker News was written in the Arc lisp dialect, a dialect created by Paul Graham. Arc was implemented on top of Racket, but that has now changed. HN runs on top of SBCL since (at least) September of 2024.

But why? For performance reasons.

> I recently noticed that Hacker News no longer uses paging for long threads. In the past, when a discussion grew large, we had to click “More” to load the next page of comments, and dang would occasionally post helpful tips to remind us about this feature. Was there an announcement regarding this change? Has anyone else still seen paging recently? I’d love to know more details—especially the technical aspects or considerations that went into the decision.

Lisp一族の
雑草のような生命力に
ご期待ください！