# スクリプト言語の概要

Alexander Kanavin
フィンランド ラッペーンランタ工科大学 教師 バーバラ・ミラフタビ、ヤン・ヴォラチェック

1 Dec 2002

**Abstract**

ここ数年、スクリプト言語への関心が劇的に高まっている。これらの言語には、従来のプログラミング言語と比較して多くの重要な利点がある。特に、コンパイルが不要であること、メモリを自動的に管理できること、高レベルのデータ型を含むことなどが挙げられる。将来、これらの言語は、既存のコンポーネントを「グルー（接着剤）」にして実用的なアプリケーション・システムにする力があるため、ほとんどのプログラミング・プロジェクトの中核になる可能性が高い。

## 1 Introduction

ここ数年、スクリプト言語は大きな飛躍を遂げた。10年ほど前、スクリプト言語は補助的なツールと見なされており、一般的なプログラミングにはあまり適していなかった。今では学術界でもソフトウェア業界でも、非常に大きな関心を集めている。

スクリプト言語と従来の言語の実行速度とメモリ消費量の比較は、[6]で研究されている。記事[4]では、スクリプト言語の歴史的背景が紹介されている。また、[1]では、商用環境におけるスクリプト言語の実用例が紹介されている。最後に、[3]では将来のトレンドが紹介されている。

この概要では、まずスクリプト言語とは何かを定義する。次に、応用分野に基づく言語の分類を示す。その後、最もポピュラーなスクリプト言語が紹介され、それぞれの言語特有の特徴が強調される。最後に、なぜスクリプト言語が重要なのか、そして将来的にスクリプト言語が果たす役割について考察する。

## 2 スクリプト言語とは何か?

スクリプト・プログラミング言語と伝統的なプログラミング言語の境界は、やや曖昧である。しかし、スクリプト言語の特徴をいくつか挙げることができる：

- インタプリタまたはバイトコードで解釈され、ネイティブコードにコンパイルされることはない。



Readable

- メモリ処理はガベージコレクタが行い、プログラマは行わない。
- リストや連想配列などの高レベルなデータ型を含む
- 実行環境は書かれているプログラムと統合できる
  - スクリプトプログラム(あるいは単にスクリプト)は、 C言語などの低レベル言語で書かれたモジュールに アクセスすることができる。

全てのスクリプト言語がこれらの機能を全て持っているわけではありません．例えば、シェルスクリプトはCモジュールにアクセスできない。しかし、それでもスクリプト言語であることに変わりはない。スクリプト言語の背後にある主な考え方は、その動的な性質であり、 データをプログラムとして扱うことも、その逆も可能である。

シェル、awk、Perl、TCL、Python、Java、Lisp、その他多数。

## 3 アプリケーション分野

The article [4] introduces four main usage areas for scripting languages:

- コマンドスクリプト言語
- アプリケーションスクリプト言語
- マークアップ言語
- ユニバーサルスクリプト言語

### 3.1 コマンドスクリプト言語

コマンドスクリプト言語はスクリプト言語の中で最も古いクラスである。1960年に登場し、プログラムやタスク制御の必要性が生じた。このような言語の第一世代で最もよく知られているのは、IBM OS／360オペレーティングシステム用に作られたJCL（ジョブ制御言語）である。

このような言語の現代的な例としては、前述のシェル言語や、sedやawkのようなテキスト処理言語がある。これらの言語は、後にPerlのような汎用的な言語にも搭載されるようになった 正規表現マッチングを直接サポートした最初の言語のひとつである。

### 3.2 アプリケーションスクリプト言語

アプリケーションスクリプト言語が開発されたのは、1980年代のパーソナルコンピュータの時代である。

その代表的なものが、マイクロソフトが開発したVisual Basic言語であり、特にそのサブセットであるVisual Basic for Applicationsは、オフィスアプリケーションのプログラミング用に設計されている。



Readable

| Name | Year of creation | Developers | Organization |
|---|---|---|---|
| Pilot | 1962 | - | IBM |
| JCL | 1964 | - | IBM |
| RPG | 1965 | - | IBM |
| MUMPS | 1969 | Okto Barnett+ | マサチューセッツ総合病院 |
| sh | 1971 | Steve Bourne | AT&T Bell Labs |
| Awk | 1977 | Alfred Aho+ | AT&T Bell Labs |
| csh | 1978 | - | UC Berkeley |
| Rexx | 1979 | Michael Qualishow | IBM UK Laboratories |
| AppleScript | 1993 | - | Apple Computer |

表1：コマンドスクリプト言語

この言語は、ユーザーインターフェイスプログラミングとコンポーネント組み込み（VBX、OCX、ActiveXなど）に重点を置いている。VBAは、以前の言語であるWord BasicやExcel Macro Languageに代わって、Microsoft Officeスイートをプログラミングするための世界共通の単一言語となった。VBAは、VBScript（OLEコンポーネントの作成とブラウザ内での作業を指向）やLotusScript（Lotus Notesプログラミング）といった後の言語に影響を与えた。

Javascript言語もこのクラスに属する。基本的には、ウェブプログラミングプロジェクトのクライアント部分の実装のためのデファクトスタンダードである。JavascriptはNetscape Navigator 2.0ブラウザで導入された。マイクロソフトのJScriptやECMAScript（ECMA-262標準）など、いくつかの方言がある。

| Name | Year of creation | Developers | Organization |
|---|---|---|---|
| HyperTalk | 1986 | - | Apple Computer |
| Visual Basic | 1990 | - | Microsoft |
| JavaScript | 1994 | - | Netscape Communications |
| CorelScript | 1995 | - | Corel |
| LotusScript | 1995 | - | Lotus Development |
| VBScript | 1995 | - | Microsoft |
| Pnuts | 2001 | Toyokasu Tomatsu | Sun Microsystems |

Table 2: Command Scripting Languages

### 3.3 マークアップ言語

マークアップ言語は、本当のプログラミング言語ではなく、テキスト文書の一部をマークアップするために使用される「タグ」と呼ばれる特別なコマンド語のセットであり、後にプロセッサと呼ばれる特別なプログラムによって使用され、ブラウザでの表示や他のデータ形式への変換など、テキストにあらゆる種類の変換を行うという意味で特殊なケースである。マークアップ言語の基本的な考え方は、コンテンツと構造を分離し、書式設定コマンドやインタラクティブなオブジェクトを文書に含めることである。

GML（Generic Markup Language）と名付けられた最初のマークアップ言語は、1969年にIBMによって作られた。1986年、ISOはGMLのアイデアに基づいてSGMLという標準を作成した。



Readable

マークアップ言語における最も有名な功績は、おそらくTeX、HTML、XMLであろう。TeXは1979年にドナルド・クヌース(Donald Knuth)によって作成され、どんなに複雑な構造の文書であっても、どのように見えるかを正確に記述するために設計された。AdobeのPostscriptとは異なり、必ずしもプログラミングの知識を必要としないユーザーを対象としている。TeXは科学界で絶大な人気を博し、複雑な数式を高品質にレンダリングするニーズを満たすことができる（他の言語ではまだ不可能である）。

HTMLは紹介するまでもなく、ワールド・ワイド・ウェブの基本言語である。HTMLは、公式のHTML標準がSGMLの観点から定義されているという意味で、SGMLアプリケーションです。(SGML自体は言語ではなく、メタ言語であり、言語を作成するための言語です)。

XMLは、簡単に言えば「SGMLの弟分」である。XMLは基本的にSGMLをより単純化、合理化したもので、データの輸送と保存、あらゆる種類のシステム間でのデータ交換に重点を置いている。複雑なデータ変換や、コンポーネントのセットアップやプログラミングなど、階層的なデータを保存するための統一的なアプローチにも使用できる。2001年、HTMLはXMLの観点から再定義され、この新しい改訂版はXHTMLと呼ばれた。

| Name | Year of creation | Developers | Organization |
|---|---|---|---|
| GML | 1969 | Charles Goldfarb+ | IBM |
| TeX | 1979 | Donald Knuth | Stanford University |
| SGML | 1986 | - | ISO |
| HTML | 1991 | Tim Berners-Lee | CERN |
| CFML (Cold Fusion) | 1995 | - | Allaire |
| DHTML | 1996 | - | Microsoft |
| XML | 1997 - | W3C | |
| XHTML | 2001 | - | W3C |

表3: マークアップ言語

### 3.4 ユニバーサルなスクリプト言語

このクラスに属する言語は、おそらく最もよく知られている。スクリプティング言語」という言葉は、まさにこれらに関連している。これらの言語のほとんどは、もともとUnix環境のために作られたものだ。しかし、その目的は異なっていた。

Perlはレポート作成のために作られたプログラミング言語であり、その名前（Practical Extraction and Report Language）にも反映されている。Perlが絶大な人気を誇る最大の理由は、この言語を使って動的なウェブページを形成するためのシンプルで効率的なCGIスクリプトを書けることだと一般に言われている。Perlは、適切な時に適切な場所に存在していたのである。

Python言語はもともと、実験的なオペレーティング・システムであるAmoebaのシステム・サービスにアクセスするためのツールとして作られた。その後、普遍的なオブジェクト指向スクリプト言語となった。Java仮想マシンや、Microsoft .NETプラットフォームで使われるMicrosoft中間言語にも実装されている。

Tcl言語は、文字列処理とTkライブラリとの密接な統合を目的として作られた。Tclは主にアプリケーション拡張言語として使われる。



Readable

PerlやPythonとは異なり、Tclは完全にスタンドアロンなプログラムを簡単に書くことができるため、CやC++の拡張モジュールに大きく依存している。

残りの言語のほとんどは、Webサービスとともに登場した第二の波に 属している。最もよく知られているのはPHPと呼ばれる言語で、HTMLと伝統的なプログラ ミング手順をループや関数と組み合わせたもので、非常に簡単にデータベースにアク セスできる。ここでは

| Name | Year of creation | Developers | Organization |
|---|---|---|---|
| Perl | 1986 | Larry Wall | - |
| Tcl | 1990 | John Ousterhaut | UC Berkeley |
| Python | 1991 | Guido Van Rossum | Stichting Mathematisch Centrum |
| Ruby | 1993 | Yukihiro Matsumoto | - |
| Euphoria | 1993 | R. Craig | これらの機能は言語に直接組み込まれており、ライブラリの一部ではない。 |
| Luea | 1994 | U. Tseles+ | PUC-Rio |
| PHP | 1995 | Rasmus Lerdordf | - |
| Mawl | 1995 | D. Ladd+ | - |
| Pike | 1996 | Frederik Hubinett | インフォメーションヴァヴァルナ |
| Curl | 2000 | Steve Ward+ | MIT Lab for Computer Science |

表4: 汎用スクリプト言語

これらの言語のいくつかは、次のセクションで詳しく説明する。

## 第4章 言語概要

### 4.1 シェルスクリプト言語

シェルはUnixオペレーティングシステムファミリーの対話型コマンドラインインタープリターです。また、if-then文、forループ、whileループ、変数、関数など、標準的な手続き型言語 の構成要素をすべて備えたプログラミング機能も備えている。

単純なシェルプログラムは非常に書きやすく自然である。しかし、プログラムのサイズが大きくなると、一種のごった煮になりがちである。また、構文の一部も非常に混乱しやすい(例えば引用のルール)。これらのルールは、対話型インタプリタとしてのシェルの主な役割を維持する試みとして導入された。

シェルプログラムは一般的に、Unixフィルタ(sort、wc、 grepなど)やawk、sedなどのテキスト処理用ミニ言語のような外部プログラムの出力の実行と処理に大きく依存しています。

シェルの主な利点は、ほとんどすべてのUnixに付属しており、インストールする必要がほとんどないことです。しかし、シェルはどんなプログラミング作業にも適しているわけではありません。

### 4.2 Perl

Perlは80年代後半にLarry Wallによって開発された。Perlは80年代後半にLarry Wallによって開発された。Perlは、awkとshellに代わるUnixスクリプトプログラミングのための言語として特別に設計されました。
Perlは、シェルよりもはるかに高度な言語であることは間違いありません。動的配列を含むより強力なデータ型や、



Readable

(電話帳のように)名前から値を高速かつ簡単に検索できる「ハッシュ」型を備えています。Perlは、テキストデータ形式のパターン駆動処理もサポートしています。これらの機能は、言語に直接組み込まれており、ライブラリの一部ではありません。
ライブラリについて言えば、PerlはUnix APIのほぼすべてを完全かつよく考え抜かれたバインディングで含んでいます。そのため、メモリ使用量やパフォーマンスを最適化する必要のない単純なタスクでは、C言語の必要性を大幅に減らすことができます。
Perlのもうひとつの強力な利点は、Perlを中心とした専用の大規模なコミュニティが育っていることだ。このコミュニティのメンバーは、グラフィカル・ユーザー・インターフェースの構築からインタラクティブな動的ウェブサイトの作成まで、事実上あらゆることを可能にする何百もの自由に利用可能なモジュールを書きました。このモジュールのコレクションはCPANとして知られている。

しかしPerlには多くの欠点がある。言語としてはきれいでもエレガントでもない。Perlの構文は一般的に可読性が低いと考えられており、最も難読なPerlプログラムを書くというコンテストが毎年開催されているほどだ。

Perlを始めるのはシェルよりも難しい。Perlはまた、プログラム設計をシンプルに保ち、プログラムサイズが大きくなってもモジュール性を維持するために、並々ならぬ努力を必要とする。先進的な言語機能の中には、既存のものの上に付け足されたようなものもある。

シェルのように、Perlはほとんどすべてのインストールで見つけることができます。Perlはマイクロソフトのプラットフォームでも利用可能であり、ドキュメントもかなり充実している。

## 4.3 Tcl

Tcl(ツール制御言語)は、コンパイルされたCのライブラリとリンクするように 設計された小さな言語インタプリタであり、Cのコードをスクリプトで制御 することができる。
Tclの上に構築されたいくつかの機能は、広く使われるようになり、 おそらく言語そのものよりも大きな人気を獲得した。これには、ボタン、ダイアログボックス、メニューツリーなどを素早く構築できる、最も使いやすいGUIツールキットの1つであるTk toolkitや、スクリプトで制御されることを意図していなかった完全な対話型プログラムを簡単にスクリプト化できる言語Expectが含まれる。Tkツールキットは Tclに限らず、例えばPerlやPythonでもよく使われている。
Tclの主な利点は、非常に柔軟でシンプルであることです。構文はやや奇妙だが(例えば関数呼び出しと'組み込み'演算子 の違いはない)、非常に一貫している。

Tclの主な欠点は、名前空間制御とモジュール性のための弱い機能しか持っていないため、大規模なプログラムを書くのが難しいことである。そのため、大規模なプログラムを書くのが難しい。
# 構文が奇妙なこともあり、初心者にとっては少々頭痛の種である。
Plain TclはUnix APIのごく一部(ファイル操作、プロセス起動、ソケットだけ)にしかアクセスできません。Tcl拡張は存在しますが、どこにでもインストールされる保証はありません。

Tclの実装はUnixだけでなくWindowsにも存在する。Tcl／Tk スクリプトは、どの実装でも変わることなく動作します。

## 4.4 Python

Pythonは、C言語と統合可能なスクリプト言語であり、プロトタイピング言語である。Tclのように、動的にロードされたCライブラリからデータをインポートしたり、Cライブラリにデータをエクスポートしたりすることができ、C言語から組み込みスクリプト言語として呼び出すことができる。



Readable

その構文は、C言語とModulaファミリーの中間のようなものであるが、ブロック構造が実際にインデントによって制御されるという珍しい特徴もある（明示的なbegin/endやC言語の中括弧の類似物はない）。これは確かにこの言語の中で最も論争の的となっている特徴だが、一度慣れてしまえば、ほとんどの人はこれを気に入っているようで、本当の時間節約になると考えている。

Python言語は、非常にクリーンでエレガントなデザインです。プログラム設計者は、オブジェクト指向のスタイルで書くか、伝統的な手続き型のスタイルで書くか、あるいはそれらのミックスで書くかを選択できる。動的リストやハッシュ配列など、Perlと似たような高レベルのデータ型が多数用意されている。また、ラムダ（単純化のために実際の関数の代わりに使える小さなインプレース関数）のような関数型言語の機能もいくつかサポートしている。PythonはTkツールキットを使ってGUIインターフェースを簡単に構築できる。

Pythonの標準配布には、ほとんどの重要なインターネットプロトコル（SMTP, FTP, POP3, IMAP, HTTP)のクライアントライブラリと、HTMLのジェネレータクラスが含まれています。このため、Pythonはプロトコルロボットやネットワーク管理スクリプトの 構築に非常に適しています。Pythonはまた、動的なCGIウェブページを書くのにも適しており、Perlと競合して、高複雑度のアプリケーション分野で成功を収めている（例えばGoogleはPythonを多用している）。

この概要で取り上げた言語の中で、多くの開発者が協力する大規模で複雑なプロジェクトに適しているのは、PythonとJavaだけである。しかし、PythonはJavaよりもシンプルで、ラピッドプロトタイピングにも適している。Java仮想マシン用のPythonの実装は、これら2つの言語の混在使用を意図しており、Jythonと呼ばれています。

他のスクリプト言語と同様、PythonはCやC++ほど高速ではありません。しかし、プログラムの重要な部分をCで書き直し、Pythonで接着することはよくある。Pythonは小規模なプロジェクトや正規表現に大きく依存するスクリプトにはPerlほど向いていない。Pythonは小規模なプロジェクトや正規表現に大きく依存するスクリプトにはPerlほど向いていませんし、小規模なプロジェクトにはやりすぎで、シェルやTCLの方が向いているかもしれません。
Perlのように、Pythonには確立された開発コミュニティがあり、Pythonのツールや拡張モジュールがたくさんあるWebサイトがある。

Pythonの実装は、Microsoft OS用とMacintosh用がある。クロスプラットフォームでのGUI開発は、Tkや他のツールキットで可能です。

### 4.5 Java

Javaは元々、ジェームズ・ゴスリング（当時は 「Oak 」という名前で知られていた）がC++の後継になることを意図してサンで開発したオブジェクト指向言語である。1993年から1994年にかけてインターネットが爆発的に普及した後、Javaはバイトコード解釈言語へと変貌を遂げ、分散アプリケーションのための新しい言語として売り込んだサンの大々的な広告キャンペーンの焦点となった。

JavaはC++よりも強力でクリーンな設計になっている。C++とは異なり、自動メモリ管理を行う。PythonやPerlのように、JavaはCのコードと統合できる。また、ドキュメントも充実している。

しかしJavaには、ウェブブラウザのプラットフォームの違いによるサポートのばらつきや、 パフォーマンスの問題、標準的なツールキット(特にAbstract Window Toolkit)との相性の問題など、多くの欠点がある。言語自体にも問題がないわけではなく、たとえばクラスの可視性ルールは不必要に複雑だ。



Readable

C++の多重継承の問題は、インターフェイス機能を導入することで回避されており、理解や使用の難易度はわずかに低い。

Javaはすでに学界で受け入れられており、次世代のIT専門家にプログラミングの基礎を教えるツールとして、多かれ少なかれPascalに取って代わって好まれている。また、ウェブ・アプリケーション・サーバー内で動作する「サーブレット」や社内開発にも多く使われている。しかし、Javaが汎用プログラミング言語としてC++に取って代わることができるかどうかは、まだ不透明である。

Javaの実装は、すべてのUnixおよびMicrosoftオペレーティングシステムで利用可能であり、すべての純粋なJavaプログラム(GUI機能を含む)のクロスプラットフォーム移植性をサポートしている。

## 4.6 Lisp

Lispが上記のすべての言語と異なるのは、命令型言語ではなく関数型言語であるという点である。基本的なデータ型として可変長リストやツリーを用い、コードをデータとして解釈し、逆にコードをデータとして解釈するという考え方に基づいている。Lispは、解釈、ガベージコレクション、演算子のオーバーローディングなど、今では当たり前の考え方が導入された言語である。

関数型言語であるLispは、メモリを自動的に管理し、従来のプログラミング言語よりもはるかにエレガントで強力である。しかし、現代の標準（CやC++の影響を強く受けている）から見ると、接頭辞記法や括弧を使った奇妙な構文を持っている。

Lispの主な問題は、Lispが実は単一の言語ではなく、言語群であることであり、 そのため現在ではLispコミュニティは深く分断されている。このため、Lispは約束されたものを提供することができず、おそらく今日広く使われている唯一の分野は、Emacsテキストエディタの動作をプログラミングすることでしょう。

そのため、特殊なファイルフォーマットやテキストデータベースを対話的に処理するタスクに非常に適している。また、テキストエディタと密接に統合しなければならないアプリケーションの構築や、ある程度の編集機能を持つテキストブラウザとして主に機能するアプリケーションの構築にも適している。電子メールやUSENETニュースのユーザーエージェントや、ある種のデータベースフロントエンドがこのカテゴリーに入る。

5 なぜスクリプト言語が重要なのか?

ほぼ10年間、最も支配的なアプリケーション・プログラミング言語はCとC++であった。しかし現在では、システム・プログラムとアプリケーションのタイム・クリティカルなカーネルを除くほとんどすべてのタスクにおいて、CとC++は明らかに長生きしている。他のほとんどのタスク（アプリケーションのプロトタイピングや、他のアプリケーションやツールのつなぎ合わせなど）は、スクリプト言語の方が適している。
CやC++の中心的な問題点は、プログラマがメモリを手作業で扱わなけ ればならず、実行環境に任せておけないことです。変数を宣言し、ポインタ連鎖リストや次元バッファを明示的に管理し、バッファオーバーランを検出または防止し、動的ストレージの割り当てと割り当て解除を行う必要があります。
これが膨大な量の複雑さとエラーの原因となっている。バッファ・オーバーランはクラッシュやセキュリティ・ホールの一般的な原因である。メモリーリークなど、特に見つけにくいバグもある。



Readable

スクリプト言語は、実行時部分(典型的にはプログラムインタプリタ)にメモリマネージャを持つことで、手作業によるメモリ管理を回避している。複数のプログラムが1つのインタプリタを共有することで、メモリ使用量を減らすこともできる。
これは一つの側面です。もうひとつは、今日のプログラムを複数の言語で書き、それぞれのサブタスクに最適なツールを選択することが理にかなっているということだ。例えば、時間的にクリティカルな部分はC言語で、データ・アクセス・ルーチンはSQLで、そしてすべてをまとめ、その上にユーザー・インターフェースを追加する接着剤はTclやPythonで書くことができる。このスタイルで実装されたシステムの全体的な複雑さは、汎用言語で1つの大きなモノリスとしてコーディングされた場合よりも低くなるだろう。

例えば、科学的なアプリケーションを構築する場合、C／C++プログラマーは効率的な数値アルゴリズムを実装することができ、同じプロジェクトの科学者は、それらのアルゴリズムをテストして使用するスクリプトを書くことができる。科学者は低レベルのプログラミング言語を学ぶ必要はなく、C／C++プログラマーは科学に関わることを理解する必要はない。

スクリプト言語の成功例は数多くあるが、ここではそのうちの2つを紹介しよう： [1]と[2]である。

## 6 The future

記事[3]では、あるトレンドが示されているが、これには私も心から同意する。基本的に、10年が進むにつれて、C++、Java、Adaのような静的型付け言語（使用前に変数宣言を必要とする言語）の使用は減少し、Python、Ruby、Perlのような動的型付け言語の使用が増加するだろう。今後数年で主流になるのはこれらの言語である。その主な理由は、コンパイル時間がほとんどゼロであること（超高性能マシンで一晩かけてコンパイルしなければならないこともあるC++とは異なる）、型安全性が不要になり開発やテストが大幅に簡素化されることである。というわけで、PythonやRubyのような言語から目を離さない方がよさそうだ。これらは非常に重要になる可能性が高い。

## References


[1] ケーススタディ： このような場合、Pythonは、MicrosoftのGreg Steinによって、第6回国際Python会議予稿集、http:／／www.python.org／workshops／1997l0／proceedings／に掲載されています。

カーネギーメロン大学のアリス・バーチャル・リアリティ・プロジェクト、http:／／alice.cs.cmu.edu

[3] Robert Martin. Are scripting languages the wave of the future? http://www.itworld.com/AppDev/1262/itw-0314-rcmappdevint/

[4] Ruslan Bogatyrov. The nature and evolution of scripting languages. http://www.osp.ru/pcworld/2001/11/144.htm




Readable

[5] Eric Raymond. To C or not to C. http://tuxedo.org/ esr/writings/taoup/chapter03.html

[6] Prechelt, Lutz; An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=ira/2000/5



Readable