Python の支援活動 HOWTO

リリース *2.7ja1*

Guido van Rossum
Fred L. Drake, Jr., editor

2011 年 12 月 25 日

Python Software Foundation
Email: docs@python.org

目次

Author A.M. Kuchling

Release 0.03

概要

一般的に、あなたの上司にオープンソースのソフトウェアの利用の許可をもらうことは難しく、Python でも例外ではありません。この文書は、なぜ Python を使うのか、利用の許可をもらう戦略、利用できる事実や議題、そして Python を使おうとする **べきではない** 場合について議論します。

1 Python を使う理由

あなたの開発プロセスにスクリプト言語を組み込む理由はいくつかあります。そして、なぜ Python がとりわけ良い選択となるのかについて議論します。

1.1 書きやすさ

プログラムはしばしば機能を組み合わせるように作成されます。下位レベルの命令はグループ化され、そして上位レベルの関数から呼ばれます。そしてその関数はさらに上位レベルの関数にとって基本的な命令として利用されます。

たとえば、ハッシュテーブルにアクセスするような非常に下位のレベルの関数セットは最下位レベルで定義されます。次のレベルではメールのヘッダ情報を保持する用途や、メールのヘッダ情報のうち `date` のような属性を `Tue, 13 May 1997 20:00:54 -0400` のような値にマッピングする用途に利用されます。メッセージオブジェクトに対するさらに上位レベルの操作では、ハッシュテーブルに保存されたメッセージのヘッダが含まれていること自体を知らずに処理しています。さらに上位のレベルも然りです。

よく、下位レベルでは単純なことをします。二分木やハッシュテーブルのデータ構造の実装や、文字列データを数値に変換するといった単純な処理です。そのため上位のレベルでは、それらの基本的な処理を組み合わせて利用します。この方法においては、これら基本的な処理は組立式の基本部品のように扱えます。そして、完成品を生み出すために組み立てて利用されるのです。

なぜこの設計手法が Python のと関係するのでしょうか。それは、Python が糊言語として最適だからです。Python モジュールを書く際によくやる方法として、命令を低水準言語で実装することです。実行速度のため、C や Java、Fortran のでさえもその候補としてあり得ます。それら基本的な処理が Python のプログラムから利用可能になってからは、基本的な上位レベルの命令は Python コードの形で書かれています。そのため上位レベルのロジックは理解しやすく、また編集しやすいものとなっています。

ジョン・オースタウトはタイトル：” スクリプティング：21 世紀の高水準プログラミング言語” という論文を書き、非常に長きにわたってこのアイデアを説明しています。この論文を読むことをおすすめします。URL は参考文献に記載しています。オースタウトは Tcl の言語の発明者で、論文の議題は Tcl でのためであることには違いありません; かれは Python,Perl,Lisp/Scheme など他の言語については軽く触れただけですが、実際のところオースタウトの議論は、先に述べた一般的などのスクリプト言語向けに誰でも拡張機能を作成できることから、どのスクリプト言語に対しても適用できます。

1.2 プロトタイプ

人月の神話 でフリードリッヒ・ブロックスは、次のことを提言しています：“捨てるつもりで作りなさい。どうせそうなるのだから”。つまりブロックスは、ソフトウェアの最初の設計は間違いであることが多いと言っています。プログラムがとても単純か、もしくはあなたがとても優秀な設計者でない限りは、開発が実際に進むにつれて新たな要件や機能が明確になってくることに気づくでしょう。もしそれらの新しい要件をプログラムの設計にきれいに組み込めなければ、うれしくない二つの選択を迫られます：何とかして新しい機能をプロ

グラムに叩き込むか、最初から作り直して組み込むかの二択です。

Python は、迅速な初期のプロトタイプ開発のために良い環境を提供してくれます。その環境よってプログラム全体の構造と処理の流れの正しさがわかり、また、Python が提供する高速な開発サイクルを通して細部を微調整することができます。いったん GUI インタフェースやプログラムの出力に満足すれば、C++、Fortran、Java といったコンパイルを伴う言語に Python を書き直せばよいのです。

プロトタイピングでは、多くの Python の機能を使用しすぎないように注意する必要があります。でないと他の言語で書き直す作業が困難になります。たとえば `eval()` 、正規表現、 `pickle` モジュール、これらを使用すると、式の評価、正規表現、直列化のために C や Java ライブラリが必要になります。しかし、これら巧妙なコードを避けることは難しいことではありません。その上、最後に書き直す場合も通常はそんなに困難な作業にはなりません。また書き直したコードは迅速なデバッグが可能です。なぜならプロトタイプの段階で重大なエラーはあらかじめ取り除かれており、あったとしても書き直しの際に取り込まれた軽微な不具合くらいだからです。

この戦略は、前述の書きやすさに関する議論に基づいています。これらのことから下位レベルの処理を接続するための糊言語として Python を利用しプロトタイプを構築ことの妥当性は明らかです。このように Python は開発の助けになります。エンドユーザーが Python コードで実装された機能に触れることがなくともです。利用する Python のバージョンが処理速度の面で妥当で、かつ企業の規定が許すならば、C や Java への書き直し作業が必要がない場合があります。そうでなくても、すぐに最終出力用の言語で書き始めるよりは、プロトタイプを作成してから書き直す場合のほうがまだ早くできます。

この開発戦略の一例に、Microsoft Merchant Server があります。バージョン 1.0 は純粋な Python で書かれています。リリース後に Microsoft によって会社ごと買収されました。バージョン 2.0 では、いくらかのコードが C++ に書き直されはじめました。結果、 C++ のコードといくつかの Python コードでリリースされました。バージョン 3.0 ではまったく Python を含んでいませんでした。すべてのコードは C++ に変換されたのです。製品に Python インタプリタがまったく含まれていないにもかかわらず、Python は開発のスピードアップする目的で有用な役割を果たしています。

これは Python にとって非常に一般的な使い方です。過去に開かれたカンファレンスで発表された論文でもハイレベルな数値計算アルゴリズムを開発するためのアプローチとして説明されています。デビッド.M. ビーズリーとピーター.S. ロムダヒさんの論文”Python で実現する大規模物理計算アプリケーション” に良い例がありますので参照してください。

もし、あるアルゴリズムにおける基本的な操作が、” この 4000x4000 行列の逆行列を出力する” ようなものである場合、またその処理がいずれかの低水準言語で実装されている時、Python によるその他の処理のコストはほとんどかかりません。`m.invert()` のような行を処理する場合に、Python を利用するがために必要になる余分な処理コストは非常に小さいです。特に、正しい処理結果を得るために途方もない微調整が必要なアプリケーションにとって良い特徴です。GUI のインタフェースや Web サイトが主な例です。

(あなたが Python に精通していれば)Python のコードは短く、速く書けます。そのため、あなたのアプローチが間違っていたと判断した場合、気兼ねなくそれを捨てることができます。もしあなたが費やした時間がわずか 2 時間ではなく二週間だったら、その 2 週間を無駄に過ごしたと認めたくない気持ちから、何とかして作ったものを取り繕おうとさらに無駄な時間を費やすでしょう。正確に言えば、それらの二週間は無駄になった訳ではありません。あなたは問題を解決するために何かを学んだはずですが、人間の本質としてそれはある種の

失敗として考えてしまいます。

1.3 シンプルさと理解のしやすさ

Python は小さなタスクにのみ使用可能なおもちゃ言語では断じて*ない*です。Python は汎用的であり、たくさんの異なった目的に使用できるほど十分強力です。10 行から 20 行程度の小さなものから、何千行もの大規模システムにスケールアップされたものにまでも有用です。

しかも、この表現力はあいまいさや巧妙さを要求するような言語の構文によるものではありません。Python はあいまいなコードにつながる記述を許す部分はあるものの、それは比較的少数であって、適切な設計によってその使用を少数のクラスやモジュールに分離することができます。たくさんの機能を、明解さを考慮せずに利用すれば、もちろん多くの誤解を生むコードになってしまいます。

しかし、ほとんどの Python のコードは人間が理解可能なように書式化された擬似コードように見えます。

エリック.S. レイモンドは、*The New Hacker's Dictionary* で、"コンパクト" について次のように定義しました:

> コンパクト ***(形)***: 設計におけるコンパクトとは、人が一度に理解できるという重要な性質を指している。一般的に、コンパクトな設計で作られた物は、コンパクトでない設計で作られた同等のものよりも、より大きなプロジェクトに利用でき、不具合をより少なく抑えられることを意味する。コンパクトさとは単に単純な構造であるとか、力の欠如を意味するものではありません。例えば、C 言語はコンパクトであり FORTRAN はそうではありませんが、C は FORTRAN より強力です。機能の増大や、全体設計にきれいに統合されていない雑な機能は、設計をコンパクトでなくします。(そのため、古い C 言語の一部のファンは、 ANSI C がコンパクトではないと主張しています)
>
> (http://www.catb.org/~esr/jargon/html/C/compact.html より)

Python はこの意味で、非常にコンパクトです。なぜなら Python はすこしのアイデアだけを元に設計されており、それが多くの場面で使用されているためです。名前空間を例に見てみましょう。`import math` 文では、`math` という新しい名前空間を作成します。一方、クラス自体も多くのプロパティを共有する名前空間です。クラスにはモジュールや自分自身のクラスも含まれます。たとえば、クラスからはインスタンスを作成できます。インスタンス・・・？実はこれらもまた別の名前空間なのです。名前空間は、現在のところ Python の辞書として実装されているため、標準の辞書型と同じメソッドを持っています。.keys（）はすべてのキーを返します。他の辞書型のメソッドも同様です。

このシンプルさは、Python の開発の歴史的なものから来ています。Python の構文はさまざまなソースから派生しています。比較的無名の教育用言語 ABC や、Modula-3 から主な影響を受けています。(ABC や Modula-3 の詳細についてはそれぞれの Web サイトを参照してください。 <http://www.cwi.nl/~steven/abc/> , <http://www.m3.org> その他の機能は、C 言語、Icon、Algol-68、そして Perl を参考にしています。Python は本来革新な言語ではありません。その替わりに、小さく学習しやすいものに維持しようとしてきました。また、他の言語で試され有用だと認められたさまざまなアイデアをもとに設計されました。

シンプルさは美徳であり過小評価すべきではありません。シンプルであればよりすばやく学ぶことができ、そしてその後すぐにコードを記述し始められます。多くの場合、最初に書いたコードがきちんと動作します。

1.4 Java での実装

あなたが Java を使用している場合は、Jython（http://www.jython.org/）に注意を払うことは確実に有用です。Jython は Java での Python の再実装で、Python コードを Java のバイトコードにコンパイルしてくれます。結果として得られる環境は非常にしっくりきており、Java とほぼシームレスに統合しています。Python から Java クラスにアクセスするのはとても簡単で、Java クラスのサブクラスとして Python のクラスを書くことができます。Jython は CPython とほぼ同じ方法で Java アプリケーションのプロトタイプ作成に使用できます。また Jython は、Java コードのテストスイートの用途にも使えます。また、Java アプリケーションにスクリプト機能を埋め込むこともできます。

2 議論と反論

あなたのアプリケーション用には Python が最良の選択だと決定したとしましょう。Python を利用することをあなたの管理者、または仲間の開発者をどのように説得しましょうか。この節では、Python を使うことに対していくつかの一般的な議論と反論を示します。

Python は無料で自由に利用できるソフトウェアです。どれほど良いことでしょうか。

非常に良いはずです。最近では Linux や Apache、二つのオープンソースソフトウエアが商用ソフトウェアの代替品として支持されるようになってきています。しかし、Python はまだ公な支持は得られていません。

Python は数年前から出回っており、多くのユーザと開発者に支持されてきました。結果的に、Python は多くの人々によって使用されて、ほとんどのバグがふるい落とされてきました。バグはまだ一定の間隔で報告されていますが、ほとんどは本当に目立たないもの（いままで誰も実行する必要がない、もしくは実行したことがないようなもの）か、外部ライブラリへのインタフェースに起因するものです。言語自体の内部は非常に安定しています。

Python では、ピアレビューを実施しながらソフトウエアを作るためにソースコードを閲覧可能にしています。だれでもコードを調べたり、改善を提案 (実装を含む）し、バグを追跡することができます。オープンソースコードの考え方についての詳細を調べるには<http://www.opensource.org>をご覧ください。オープンソースに関連した議論と、ケーススタディなどがあります。

だれが Python をサポートするのか？

Python には、かなり多くの開発者のコミュニティがあり、数はまだ増え続けています。Python に関するインターネット上のコミュニティは活動的です。これは、Python を利用する上での別の利点の一つと考えられています。comp.lang.python のニュースグループに投稿されたほとんどの質問はメンバのうち誰かによって素早く返答されます。

あなたがソースコードを読み進める必要がある場合、明解でうまく組み立てられたものであることが分かるでしょう。そのためあなた自身の手で拡張機能を記述してバグ追跡するのは難しいことではありません。Python の金銭的なサポートしている企業や個人もいます。

誰が Python を大々的に利用していますか？

Python の興味深い点は、多くの人が驚くべき多様なアプリケーションに利用していることです。Python は以下のアプリケーションに利用されています:

- Web サイトを運用する
- GUI インターフェースを書く
- スーパーコンピュータ上で複雑な計算をコントロールする
- Python インタプリタを埋め込むことによって、商用アプリケーションでスクリプティングを可能にする
- 大規模な XML データセットを処理する
- C または Java のコードに対するテストスイートを構築する

あなたのアプリケーションドメインが何であれ、きっと誰かが同じように Python を使っているはずです。しかし、そのようなハイエンドなアプリケーションに使用可能であるにもかかわらず、小さなタスクに使用できるほどに簡単です。

その他の組織的な Python の使用例については <http://wiki.python.org/moin/OrganizationsUsingPython> を参照してください。

Python の使用に関しての制限は何ですか？

制限は事実上存在しません。`Misc/COPYRIGHT` ソースの配布、または *history-and-license* 言語全体の節を参照してください。しかし結局のところ下記の三つの条件に要約できます：

- 作成したソフトウェアに著作権表示を残す必要があります。もし製品に Python のソースコードが含まれていない場合、説明書に著作権表示を入れる必要があります。
- あなたの製品は Python を開発している機関から支持されていると主張しないでください。あらゆる方法においてです。
- 何か問題が発生した場合でも、損害賠償訴訟を起こすことはできません。実質的にすべてのソフトウェアライセンスは、この条件が含まれています。

ただし、Python を含んだり、ともにビルドされるソースコードすべてを公開する必要がないことに注意してください。また、Python インタプリタとそれに付随するドキュメントは、好きな方法で変更を加えたり再配布することができます。誰かにライセンス料誰を支払う必要はありません。

なぜ我々は、明解な言語 X の替わりに不明瞭な Python を使用する必要がありますか？

私はこの HOWTO と、最後の節に記載されている資料が、Python は不明瞭ではなく、また健やかに成長しているユーザー基盤を持っていることを納得させる助けになることを願います。アドバイス：常に Python の肯定的な利点を提示するのではなく、言語 X の欠点にも注目してください。人々はなぜ他のすべての解決策が悪いのかの理由よりもむしろ、なぜその解決策が良いのかを知りたがります。そのため様々な理由で競合する解決策を攻撃するよりかは、単にどのように Python が解決できるかを示すほうが良いです。

3 有用な資料

http://www.pythonology.com/success 「Python サクセスストーリー」は、Python の利用に成功したユーザーのからの成功例のコレクションです。ビジネスと企業ユーザーを重視したものになっています。

http://www.tcl.tk/doc/scripting.html ジョン・オースタウトはスクリプト言語の有用性について議論しています。自然な流れとしてスクリプト言語としては彼が開発した Tcl を取り上げています。しかし、議論のほとんどは、その他のスクリプト言語に適用できます。

http://www.python.org/workshops/1997-10/proceedings/beazley.html 作者のデイビット.M. ビーズリーとピーター.S. ラムダヒは、ロスアラモス国立研究所での Python の利用について述べています。これは、Python は実際の作業に役立つことができる良い例となっています。下記の論文からの引用文は多くの人々によって繰り返し引用されています：

> もともとは超並列処理システムのための大規模なモノリシックアプリケーションとして開発として開発を始めました。私たちはアプリケーションをより柔軟、再利用可能、そしてより強力なもの、そしてシミュレーション、データ解析、可視化ができるものに進化させるために Python を使用してきました。

加えて、Python は開発に関連する重要ないくつもの問題を解決してきました。それは、学術ソフトウエアの、デバッグ、導入、そしてメンテナンスを含みます。

http://pythonjournal.cognizor.com/pyj1/Everitt-Feit_interview98-V1.html アンディ・フェイトとのインタビューです。infoseek での Python の利用に関する議論では、Python を選択すると、社内での開発プロセスに大して余計な問題を持ち込まないことを示していおり、さらにいくつかの重要な利点を提供できることを示しています。

http://www.python.org/workshops/1997-10/proceedings/stein.ps 第 6 回目の Python カンファレンスでグレッグ・ステイン氏は、Microsoft で、Python を利用した eShop と呼ばれるサイトの立ち上げとのその後について追跡した論文を発表しました。

http://www.opensource.org 管理者は、有料でないソフトウェアの信頼性と有用性について疑問の念を持っているかもしれません。このサイトでは、オープンソースソフトのほうがクローズソースソフトと比較してかなりの利点を持つことについての議論を提示しています。

http://www.faqs.org/docs/Linux-mini/Advocacy.html Linux での支援活動 mini-HOWTO は、この文書の着想であり読んでおく価値があります。Linux や Python のような、新しい技術を管理者に受け入れてもらうことについて、一般的な推奨事項について記載されています。一般的に、これは多くの場合、負け犬の遠吠えを言っているだけのように見られてしまいます。そうではなく、Python は他の多くの分野のシステムの改良版であるという点を指摘するほうがよいでしょう。