# PyPy: Implementing Python in Python

***Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community***

## Building a Better Open Source Programming Language

Python is an open source programming language that has become popular because it is very easy to program with and because it runs on a wide variety of platforms from mobile phones to high end super computers. Over the years, the open source community had started a variety of development threads to improve the implementation of the language. The community took the unprecedented opportunity of an EU research initiative to bring the threads together and focus the community's resources. After two years of concentrated development, the project will produce a new implementation of Python that is measurably more efficient, more flexible and useful to more end users.

## Rethinking Language Implementation

Current language implementations are static and hard to modify by users. Even the implementations of open-source dynamic languages have non-flexible designs crafted by a small group of developers. While designing a good programming language is no easy task, often application developers seek more support and configurability of their language. This is especially the case for the rising number of specialized runtime environments where small memory footprints, concurrency or real-time features are essential.

High-level languages such as Python offer a highly productive development tool. However, whole-program static compilation is often not suitable, so that such languages are usually implemented by introducing a bytecode and interpretation level, resulting in a loss of performance. This is in contrast to lower-level languages such as C, which offer performance but greatly decrease productivity.

The architectures of current interpreter implementations are a compromise between initial simplicity, portability and performance in a single code base. The resulting monolithic code bases are inflexible and expensive to evolve. Early design decisions concerning aspects such as memory management, object model and layout, security, multi-threading model are deeply tangled in the code and cannot be reverted, experimented with or adapted to specific runtime environments.

In the long run, user pressure tends to shape interpreters towards performance at the expense of other aspects (simplicity, flexibility), possibly culminating in the introduction of a native 'Just-In-Time' (JIT) compiler, adding a significant amount of code and complexity and further impairing the flexibility.

Consequently, application developers are often left with bad choices not only regarding productivity versus performance, but they have to work around hard-wired characteristics built into the languages. Instead they would like to adapt and configure a high-level language to suit their needs while at the same time developing on top of a custom but standardized language offering both productivity and performance. For the first time, PyPy will offer them that.

***Making Python so good you will never need to use another programming language.***

## Making Python Better

The aim of the PyPy project is to research and implement an interpreter and runtime environment for the Python language, built on a unique architecture enabling both productivity and performance. The main building block is the concept of an *Object Space*, which cleanly encapsulates computational operations on objects, separated from the bytecode interpreter. A number of features, most of which are hard to implement as application-level libraries, can become part of the language in a manageable and user-configurable way if they are implemented modularly as Object Spaces. For example:

- Choice of memory and threading model
- Choice of speed vs. memory trade-offs
- Distributed/parallel execution (SMP or Networked)
- Orthogonal persistence

### Benefits of PyPy

- *More efficient* Python programming language.
- An implementation that is *more easily portable* to new architectures such as *embedded systems* and *mobile phones*.
- A Python language that can be used naturally in *more application domains* such as *systems programming*, *simulation*, *games*, *optimisation problems* and the *semantic web*.

- Pervasive security support
- Logic and aspect-oriented programming

The second key idea is to write the interpreter and Object Spaces in the Python itself and recover performance with a separate translation process producing specialized efficient low-level code. The translation is not one-shot: it can be repeated and tailored to weave different aspects into releases with different trade-offs and features, giving the user of the language a real choice.

Apart from work on translation and interpreter architecture, PyPy will improve the suitability of Python for programming tasks in the areas of multi-threading, aspect oriented programming, constraint-based optimisation and search.

## Innovation in Language Implementation, Programming Tools and Software Development Practices

PyPy will use a novel object oriented interpreter architecture based on the *Object Spaces*. It will also use **a**bstract interpretation of the *same* Bytecode interpreter with a different Object Space to give unprecedented power and simplicity for all kind of source analysis tools. In this way, the project should narrow the gap between a flexible, high-level language such as Python and more static low-level languages such as Java and C.

The use of object spaces, abstract interpretation and translation of much C code in the current Python implementation to Python itself offer unprecedented flexibility to the language user. This will be supported with a new suite of tools to allow users to customise the translation process and the interpreter that will eventually run their programs.

Another innovative side to PyPy is the way it has adapted ongoing open source projects with typical open source development practices to the demands of EU contract-based funding. This could prove a useful source of experience for other commercial and public sector bodies that would like to make use of the development potential of the open source community.

## Potential Impact

The PyPy implementation should quickly reach the large user base of the current, industrial-strength Python, and could eventually form the foundation of the “next generation” Python implementation commonly referred to as Python3000.

Moreover, PyPy offers a practical and scalable approach to interpreter modularity, applicable to any language, from general to domain-specific ones In addition. Thus PyPy expects to have an impact on communities not currently using Python, through the following:

- The build tools to generate customized versions of Python, attractive to a larger industrial base which is currently outside the scope of Python for reasons such as performance, configurability, or end-user device resources;
- The framework of PyPy's source code, reusable to implement other general or domain-specific languages;
- The theoretical approach and solutions that will be published in the relevant scientific forums.

*Sprints: Regular open-invite coding work-shops form the key to PyPy software development.*

| | |
|---|---|
| Project Reference: | 004779 |
| Start Date: | December 1, 2004 |
| Duration: | 2 years |
| Project Cost: | € 2.3 Mio |
| Contract Type: | STREP |
| End Date: | November 30, 2006 |
| Project Funding: | € 1.3 Mio from EU |

URL: http://codespeak.net/pypy/
Contact persons: Alastair Burt (burt@dfki.de)
Stephan Busemann (busemann@dfki.de)