Why Python?

Written by Kundan Singh
Sunday, 21 September 2008 11:00 - Last Updated Friday, 01 August 2014 16:26

Python encourages developers' efficiency because of very compact, concise (and beautiful!) software that one can write. Based on my experience the number of lines-of-code is 5 to 30 times less than the corresponding code in Java or C/C++.

I have written SIP stack four times now in four different languages. First was a modification of a C stack based on RFC 2543 in 1999-2000. Later I reworked the user agent API and the library to create a RFC 3261 based C++ stack in 2002, with some C code hanging around. After graduating and joining a company, I wrote another SIP stack in ActionScript, which is like ECMAScript, in 2006. And finally I wrote again for this project in Python end of 2007. The first time it took around 4-5 months to get things going in C, then around 3-4 months for C++, around one month in ActionScript and about a week in Python. Based on my experience, generally for every line of Python code there is about 3-10 lines of ActionScript code, 5-10 lines of Java code, and more than 20 lines in C.

The source code for RFC 3261 implementation in Python for user agents is about 1300 lines, whereas a similar library in C++ or Java extends to more than ten thousand lines of source code. For a significantly large project, if designed right, one can achieve a factor of improvement in lines-of-code. Clearly writing one line of Python code takes more time than writing one line of Java code, but reducing the overall lines-of-code has many fold advantages:

1. Less number of lines means that one can write software much faster. Implementing a typical RFC takes a day or two, unless it is big like RFC 3261 (SIP) in which case it may take a week or so. On the other hand, a C++ or Java programmer will have to spend much longer to get things going.
2. There is less garbage (syntactic sugar) in the source code, which means reading and understanding the code is easier. Syntax highlighting tools, and embedded docstring mechanism help in a number of ways.
3. Less code means that the testing and review efforts are less. The embedded doctest style testing is pretty convenient for small routines. Lot of code is easily reviewed just by reading the code and the corresponding specification or documentation, unlike in C or Java where there could be too much of interdependencies among different classes, methods and functions.
4. Less code means there would be less number of bugs.
5. The improvement in programming efficiency has further effect on the programmer, as he gets more motivated to write more code, instead of getting stuck in dealing with lots of code. While I don't have the numbers, I would guess that the development and testing cost in terms of man-hours asymptotically grows faster than the number of lines of code.
Why Python?

Written by Kundan Singh
Sunday, 21 September 2008 11:00 - Last Updated Friday, 01 August 2014 16:26

I have written many large software pieces in many different programming languages, C, C++, Tcl, Perl, Java, ActionScript and Python. And I have come to a conclusion that Python is probably the best, and should be used for all applications if possible. Projects where Python may not be used are those which have specific constraints such as low level access or performance requirements (C/C++), or need to work within a browser (JavaScript) or server (PHP).

Using the Python programming language helps us achieve the following objectives and guidelines in our software. The implementation is largely compatible with Python 2.7 (not with 3.x), but may be ported in future to other languages.

- **Portability:** apart from the Python standard library, the project should not rely on other third-party libraries. If such third-party libraries become necessary, consider including them in the repository or provide clear instructions for such dependency. The module should isolate such dependencies to smaller part if possible. The project should therefore be portable to many interpreters and runtime environments.

- **Threading:** threading vs event-driven programming style is decision that best left to the application developer instead of forcing a particular choice in the library. The project should not impose such decision in any reference implementation. If it is necessary to include such choice, then it should provide reasonable set of alternatives pre-built in the module.

- **Concise:** Python enables expressing ideas in code in less number of lines. The programmer should further honor the Pytonic programming style. Less number of lines means that one can write software faster, and with less garbage (syntactic sugar), one can read and understand the code easily. Moreover, testing and review efforts are less. The resulting improvement in programmer's efficiency reflects in her motivation to write more clean code.

- **Testing:** testing is an integral part of all code in this project. It uses doctest wherever possible to integrate code with documentation and testing. Alternatively, dedicated test and sample applications are included for manual or automated testing. Generally, running a module on command line via Python interpreter should run its test cases and report any errors.

- **Logging:** standard logging module is used at various log levels. The code should avoid using the standard print statements as much as possible.
for logging - helps in migrating to Python3 in future, and reduces unwanted output when the module is included elsewhere.

For completeness, following non-goals are documented too. This project does not address and does not attempt to achieve the following non-goals.

- Performance: when high performance is desired, it is recommended to either use another project or port portions of this software into C/C++. This project does not attempt to improve the performance if it hurts the readability of the code.

- Non-standard or custom extensions: while the goal of this project is to provide implementations of standards and sample applications, many real-applications often need extensions that are not yet standardized or are proprietary but popular, or need customization such as to interact with other software or user interfaces. Such extensions can easily be added by third-party developers or via support from our business unit.