PYTHON: THE PROGRAMMER’S LINGUA FRANCA

USING PYTHON TO BUILD BRIDGES BETWEEN TECHNOLOGIES, BUSINESSES AND PEOPLE
There are a multitude of programming languages in use today; dozens of very popular languages with wide user bases, hundreds in niche applications, and more emerging every year. Some are more specialized, some are general purpose. Each has its strengths and weaknesses, but the huge variety of languages in use makes for a confusing technological landscape.

The desire to optimize code for specific kinds of business or technical problems is one of the factors that has led to the enormous proliferation of specialized programming languages.

Businesses have to choose programming languages for the projects at hand by weighing the advantages of using a specialized language against the difficulty in finding developers with proficiency in a potentially uncommon language.

Programmers have a related dilemma: invest time learning a popular, general-purpose programming language to open up a wider range of employment opportunities, or specialize in a niche language which has fewer, but possibly better paying opportunities.

It's not an all-or-nothing game for either party though. One path out of the conundrum is to use the specialized language where appropriate, and a general purpose language to connect with other software and communicate with developers in the industry at large. Ideally, this general purpose programming language would be comprehensible to everyone and able to interact with as many specialized languages and technologies as possible. A common language.

This paper presents the case that Python is the language best suited to becoming the new programmer’s lingua franca.

WHAT IS A LINGUA FRANCA?

A lingua franca is a common language used for communications between people who do not share the same native language. The historical “Lingua Franca” was used throughout the Mediterranean for commerce during the Renaissance. Though made up mostly of Italian, its vocabulary included words from Arabic, French, Greek, Portuguese, Spanish and Turkish. If you wanted to do business across the Mediterranean in the 1600s, you used lingua franca or you dealt with a lot of translators.

Today, English is considered the lingua franca for international business, aviation, science, and high tech. Attempts have been made to formalize this type of English usage with several “simplified English” models, such as Basic English. Its limited vocabulary makes it easier to learn for non-native speakers by cutting out most of the notoriously large vocabulary that Standard English has accumulated.

“There are some technical advantages for having programming languages match the vocabulary of various problem domains. For one thing, such languages are easy to learn for programmers who are familiar with the various domains.”

- SOFTWARE ENGINEERING BEST PRACTICES: LESSONS FROM SUCCESSFUL PROJECTS IN THE TOP COMPANIES, CAPERS JONES
Programming languages don’t operate in the same environment as human ones, and they don’t evolve the same way. They are artificial constructs that need to be “understood” by interpreters and compilers as well as humans, but the need for a common working language is the same.

OTHER CONTENDERS

The need for a common programming language predates Python, and there are a few other languages which have been ubiquitous enough to fulfil the role.

BASIC

Though not now in widespread use, many programmers working today started with BASIC during the first wave of home and kit computers. For many years, it was the de-facto first language of hobbyist programmers, and many would argue that the micro-computer revolution could not have happened without it.

There were many implementations of the language, by necessity. With no standardized computer architecture or operating system, BASIC was a common thread that linked together amateur programmers working on a variety of first-generation microcomputers. Programs were published as source code in magazines and the language used was most often BASIC.

But BASIC is no longer the common language of programmers. Even though language descendents like VisualBasic .NET are still extremely popular (used by more programmers, if not a larger percentage of programmers), they do not play the same important part in programming culture that BASIC did in its day.

You all know C right?

When programmers need to demonstrate concepts using actual code, the primary criteria for which language to use is “What’s the language that most of my audience will likely know?” C and Java have traditionally been the ones chosen.

It’s hard to fault C in this role. It’s the most widely used programming language of all time, compilers are available for almost all platforms and operating systems, and it has been an important part of most computer science curriculums for decades.

“Meanwhile, the same guys were offering ... a programming language that folks could use to create new software of their very own. BASIC was derived from academic research tools like beloved old FORTRAN ... It was crude. It was dry. It was unsuitable for the world of the graphic user interface. BASIC had a lot of nasty habits. But it liberated several million bright minds to poke and explore and aspire as never before.”

- WHY JOHNNY CAN’T CODE, DAVID BRIN

“Although C is becoming increasingly rare, it is still the lingua franca of working programmers. It is the language they use to communicate with one another, and, more importantly, it is much closer to the machine than “modern” languages that you’ll be taught in college like ML, Java, Python, whatever trendy junk they teach these days.”

- ADVICE FOR COMPUTER SCIENCE COLLEGE STUDENTS, JOEL SPOLSKY
However, what Spolsky refers to as “trendy junk” is a reality for current students, recent graduates, and anyone working in web development. Though C might be an important part of a well-rounded and complete computer science education, many programmers will never need to work with lower-level languages like C. Higher level interpreted languages offer a much easier, and often more concise, way to build applications. The performance concerns that would normally favor a compiled language are gradually being addressed by improved interpreters and Moore’s Law.

For programmers without a background in C, it’s syntax can seem idiosyncratic. This is not a serious problem for C programmers, or those willing to invest the time to become C programmers, but for those simply trying to understand a code sample it can be a significant barrier.

JAVA

Java is widely used as a computing lingua franca for some of the same reasons as C: it’s extremely popular (vying with C for top spot depending on which index you consult), widely taught in computer science programs, and cross-platform.

Unfortunately, Java is a verbose and repetitive language. It often takes a lot of code to describe even simple algorithms or concepts. Writing the code can be manageable if you have a good IDE to help fill in class declarations and boilerplate, but it’s a pain to read, especially for the uninitiated. If your application or web service only has a Java programming interface there will be a large body of people who can use it, but for those who don’t know the language, learning enough Java to do anything useful is a lot to ask.

What we actually need is a language that’s not just popular, but easy to understand for those that don’t already know it.

WHY PYTHON IS BETTER

In recent years, the use of Python for sample code and programming interfaces has increased. It is taking over as the general purpose language used to demonstrate concepts, provide a common vocabulary, and to glue together systems created in other languages.

So what is it about Python that driving its adoption as a lingua franca?

“This emphasis on readability is no accident. As an object-oriented language, Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it’s not readable. Many of Python’s features, in addition to its use of indentation, conspire to make Python code highly readable.”

- INTRODUCTION TO PROGRAMMING PYTHON, GUIDO VAN ROSSUM
Ease of Learning, Ease of Use
Python was designed to be easy to learn, easy to understand, and most of all easy to read.

With readability and re-usability at the core of the language’s design, one would expect that becoming proficient in the language would be easier. Though it’s not something that’s easy to quantify, the experiences of veteran programmers moving to Python from other languages seem to indicate that this is the case.

“... I was generating working code nearly as fast as I could type. When I realized this, I was quite startled. An important measure of effort in coding is the frequency with which you write something that doesn’t actually match your mental representation of the problem, and have to backtrack on realizing that what you just typed won’t actually tell the language to do what you’re thinking. An important measure of good language design is how rapidly the percentage of missteps of this kind falls as you gain experience with the language.

When you're writing working code nearly as fast as you can type and your misstep rate is near zero, it generally means you’ve achieved mastery of the language. But that didn’t make sense, because it was still day one and I was regularly pausing to look up new language and library features!

This was my first clue that, in Python, I was actually dealing with an exceptionally good design. Most languages have so much friction and awkwardness built into their design that you learn most of their feature set long before your misstep rate drops anywhere near zero. Python was the first general-purpose language I’d ever used that reversed this process.”

- WHY PYTHON?
ERIC RAYMONDS

WHAT DOES GOOD LANGUAGE DESIGN LOOK LIKE?

The following three code samples compare a minimal class in C++, Java and Python. The code was pared down from working examples[1] implementing a tree algorithm.

- The following elements are common to all three:
  - Comment blocks
  - Class definition
  - Constructor method definition
  - Public methods with arguments
  - Class level attributes
  - Object instance attributes
  - Attribute get and set operations
  - Assignment, conditional and return statements
  - Object pointers, integers and null values

C++:

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

template<class t> class redblacktree {
private:
  static const int red = 0;
  static const int black = 1;
  int m_color;
  t m_val;
  redblacktree *m_left;
  redblacktree *m_right;

  redblacktree(redblacktree *b) {
    m_val = b->m_val;
    m_left = b->m_left;
    m_right = b->m_right;
  }
};
```
m_color = red;
}
public:
  redblacktree(t x) {
    m_val = x;
    m_left = 0;
    m_right = 0;
    m_color = red;
  }
  const redblacktree *find(const t &key) const {
    const redblacktree *result = 0;
    if (key == m_val) {
      result = this;
    }
    else if (key < m_val) {
      if (m_left != 0) {
        result = m_left->find(key);
      }
    }
    else {
      if (m_right != 0) {
        result = m_right->find(key);
      }
    }
    return result;
  }
};

Java:

import java.util.*;
public class redblacktree<t extends comparable<t>> {
  public static final int red = 0;
  public static final int black = 1;

  private int __color;
  private t __val;
  private redblacktree<t> __left;
  private redblacktree<t> __right;

  private redblacktree(redblacktree<t> b) {
    __val = b.__val;
    __left = b.__left;
    __right = b.__right;
    __color = red;
  }
  public redblacktree(t x) {
    __val = x;
    __left = null;
    __right = null;
    __color = red;
  }
  public redblacktree<t> find(t key) {
    redblacktree<t> result = null;
    if (key == __val) {
      result = this;
    }
    else if (key.compareTo(__val) < 0) {
      if (__left != null) {
        result = __left.find(key);
      }
    }
    else {
      if (__right != null) {
        result = __right.find(key);
      }
    }
    return result;
  }
}
Python:

class redblacktree:
    red, black = range(2)
    def __init__(self, val=none):
        self.left = none
        self.right = none
        self.val = val
        self.color = redblacktree.red
    def find(self, key):
        result = none
        if (key == self.val):
            result = self
        elif (key < self.val):
            if (self.left != none):
                result = self.left.find(key)
            else:
                if (self.right != none):
                    result = self.right.find(key)
        else:
            if (self.right != none):
                result = self.right.find(key)
        return result

The first major difference is overall size. Python is half the size of Java and about 40% the size of C++. This is common.

The next thing you’ll notice in Python is the absence of setup and typing declarations. The algorithms here are all the same, but Python gets right to the problem at hand.

It is conceivable that the Java and C++ could actually be generated from the Python code. This approach is the essence of Jython. Using the Python language doesn’t necessarily mean using a specific Python interpreter or compiler.

OPENNESS

English could not have become a global language if you had to pay to use it. The reference implementation of Python (AKA CPython) is both free and open source. Under the terms of the Python Software Foundation License, users can “reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python” so long as the copyright notice is preserved.

Vendor supported interpreters like ActivePython are available, but no single company can restrict the development of the language itself or the interpreter it runs on, as was the case until quite recently with Java. With Python, there have never been any such restrictions, and a number of competing alternative implementations[2] have evolved to fill specific niches.

PYTHON AS A SECOND LANGUAGE

Python has a user and contributor community which is more evangelical than most. The creators of Python and the community of users are clearly proud of its design and actively promote it. This translates into a wealth of resources for learning the language, many available for free. Python itself has excellent documentation and tutorials and the Python wiki has a comprehensive list of books, tutorials and other learning material in a wide range of (human) languages[3]. Not speaking English is not a barrier to learning Python.

More than any other language, Python also excels at introducing itself to people who are already programmers in other languages. This is a testament to the number of people who have moved to Python from other languages, and there are helpful guides on moving to Python from Java and C specifically.

Google has made their own internal Python training program available online[4], and ActiveState offers customized training for enterprises introducing Python to their development teams.

INTEROPERABILITY AND EXTENSIBILITY

Some applications and ideas are universally useful. A programmer working with one programming language
THE PROGRAMMER’S LINGUA FRANCA

should be able to communicate a technique or an algorithm to a colleague working with another. If their applications could talk to each other directly, regardless of the differences in their technology stack, that would be even better. Ideally, code itself could actually be shared and used verbatim.

Since there are so many programmers in the Python community with backgrounds in other languages, there are a large number of technical bridges back to those other languages. These bridges can take a few forms:

› Python interpreters implemented in the target language or technology - e.g. Jython is a Python interpreter written in Java; IronPython is written for the .NET framework.
› CPython embedded in a C/C++ application - used by many applications to provide a scripting interface[5]
› Modules which provide language parsers/interpreters or connectivity to the target language - JPype gives Python programs access to Java class libraries; RPy provides an interface to the R programming language

There are thousands of contributed modules available from the Python Package Index (PyPI) and ActiveState’s PyPM package repository. A number of these are dedicated to providing connectivity to databases (Oracle, SQL Server, DB2, SAP DB, Sybase, MySQL, PostgreSQL, SQLite, redis, MongoDB) or parsing for open data formats (JSON, YAML, and XML).

HOW TO MAKE PYTHON WORK FOR YOU

If you have a new project which could benefit from the simplicity, openness and extensibility of Python, you have a clear starting point. But even if you are currently working with another language, there are still a number of opportunities to use Python in conjunction with your existing software or to expand your personal knowledge of the language for future projects.

In an environment where software providers often try to herd customers into a single technology stack, Python excels at bridging the gaps between technologies, keeping options open to use whichever solution is right for the task at hand.
HOW ACTIVESTATE CAN HELP

ActiveState has been helping individuals and organizations use open source dynamic languages for over a decade by removing the barriers to adoption. For programmers, this means powerful development tools for Python like Komodo IDE, and the up-to-date and easy to use ActivePython distribution for multiple platforms (Windows, Mac OS X, Linux, Solaris, HP-UX, AIX), the PyPM binary package manager, and community resources such as the ActiveState Code site. For IT or development managers looking to introduce or expand Python usage in their organization, ActiveState has several levels of commercial support: development advice, on-site Python training, IP indemnification, and OEM bundling re-distribution licenses.

ActiveState, the open source languages company, believes that enterprises gain a competitive advantage when they are able to quickly create, deploy and efficiently manage software solutions that immediately create business value, but they face many challenges that prevent them from doing so. The company is uniquely positioned to help address these challenges through our experience with enterprises, developers and open source technology. ActiveState is proven for the enterprise: more than 2 million developers and 97 percent of Fortune 1000 companies use ActiveState’s end-to-end solutions to develop, distribute, and manage their software applications written in Perl, Python, Ruby, Go, Node.js, Lua, Tcl and other dynamic languages. Global customers like Cisco, CA, HP, Bank of America, Siemens and Lockheed Martin trust ActiveState to save time, save money, minimize risk, ensure compliance, and reduce time to market. To learn more visit, ActiveState.com.

© 2017 ActiveState Software Inc. All rights reserved. ActiveState, ActivePython, and ActiveState Komodo are registered trademarks of ActiveState. All other marks are property of their respective owners.