

So, I've decided to learn a new programming language, and to use this programming language for our QA/QC/KitchenSink project. What language was chosen, and why? Python was chosen. The why is a little bit longer.

### **Requirements (of the language itself)**

- Cross platform
- Wide developer knowledge
  - Easier to go for help
  - Communities are easier to find and integrate into
  - Easier to find someone to hire later for maintenance
- Fast (or as the line goes, "Fast enough")
- Preferably can compile to byte code to save the compilation on each invocation of the program
- Object Oriented
  - Preferably in a way that is intuitive and fluid. That is, the language was designed from the ground up to be OO; it is not something that was added to the language later.
- Dynamic (i.e. not statically typed)
  - Makes dealing with databases and processing variable/dynamic data much easier
- Strong code introspection
  - Allows run time extraction of information about code.
  - Code introspection makes it easier to implement code that is loaded and executed at run time, thus making things like plugins and add-on data tests much easier to implement, as the code itself can carry the information about the actions it needs to carry out.
- Clean syntax
- Automatic garbage collection
  - Greatly reduces the amount of time needed for debugging and eliminates memory leak bugs

### **Requirements (functional needs)**

- Mature, fast, aesthetically pleasing graphing package, or bindings for said package
- GUI Tool kit bindings
  - This also entails an easy-to-use GUI designer so time is not spent manually coding up how the GUI looks, but instead is spent designing the GUI visually and coding up the actions performed by the GUI.
- Mature, easy to use ORM (Object/Relational Mapping) packages
- Mature IDE
  - Cross platform (or in our case, at least runs on Linux)
  - Easy to use, graphical debugger
  - "Intellisense" The Microsoft term for popping up lists of objects or arguments when you hit "." on an object or "(" to start a function call. I got hooked on/spoiled by this feature (for better or worse) way back when I started on VB in MS Access 97: It's one of the reasons why I learned VB so fast. And when learning a new language, or using modules or features with which I'm not familiar, it greatly increases the speed at which I learn and am able to use a larger part of the capabilities available to me. It also keeps one from making so many trips to the documentation when one is trying to remember the name of that function/method/etc one needs.
  - Free
- Strong templating language (for web page generation)

Side note about GUI's and ORM's. The reason I want the language to be both a good web language and a good GUI/desktop language is it will reduce the logistical burden. The ORM can be designed once, and used on both sides (GUI and web), and only has to be maintained in one place.

## The Contenders

I won't go through every point above for each language, but mainly hit the reasons why the languages were not chosen.

### C/C++

While an "old standard," C/C++ just has too much overhead for the GUI and web software we want to turn out. It is certainly not dynamic, which makes writing plugins very tedious, and of course there is *no* garbage collection, with all memory allocation and cleanup being done by the programmer. Programming with Qt helps in some of this, as you can have parent objects clean up their children, but by and large, careful attention must be paid to memory, pointers, and the like. Also, ORM's are almost non-existent, as defining objects and behaviors at run time is difficult, if not impossible in a static, compiled language.

### PHP

PHP is a good language for web based projects, but at the same time gets very tedious for large projects. It also is not very conducive to separating content from presentation, as code and HTML often gets very mixed. To be very honest, I have not investigated PHP IDE's or ORM layers. While PHP does have GUI bindings, it is not a language in which I want to write a GUI. In addition, due to its embedded-in-web-pages nature, syntax in PHP can be *very* haphazard and ugly. PHP does have a strong developer community, and it is fairly easy to the skill in new hires.

### Perl

I have programmed in Perl for several years, and overall, I like the language. The main strikes against Perl are:

- A lack of mature bindings to Qt (the GUI tool kit of choice)
- Lack of a good IDE for Linux. All of the IDE's (save an Eclipse plugin) for Perl I have seen have been for Windows. Since both of our developers are now running Linux, this won't work. In addition, the good Windows-based IDE's I did find were not free.
- Perl OO is a bit of a bolt-on type addition to the original language, and writing and using classes in Perl can feel a bit clunky at times.
- No graphical debugger for Linux (see comment about IDE's)
- Perl has been called "compilable line noise." Syntax in Perl can be ugly if one wants, or very pretty if standards are adhered to. In a large project, though, combing through all the %, \$, @, and -> can be very tedious on the eyes.
- The best I've found for graphing in Perl is using the GD bindings.

### Ruby

I've evaluated this language very little. It does have some very strong points, such as being totally OO, clean syntax, GUI tool kit bindings, strong ORM (can we say Ruby on Rails/ActiveRecord?) and good IDE's (I think). Despite these up sides, it is still a new language so does not yet have the developer uptake that older languages would have, nor does it have the breadth of modules and libraries that we would need (e.g. graphing), although both of these things are improving.

### Python

As was stated at the beginning, my final choice was Python. In fact, the list of the first page was in large part a list of Python's features and benefits against which to measure other alternatives. So, instead of restating what I already did, I can just say that Python fulfills all the requirements listed above, and will be a strong choice for our QA/QC system: the GUI front end, the web page generation, and the actual data testing/evaluation.