From: www.itworld.com

Which language is right for you?

by Cameron Laird and Kathryn Soraiz

March 15, 2001 —Pick the language with which you're most comfortable. That's our advice to reader John Wiersba, who asked us several weeks ago where he should concentrate his study.

Selection principles

For the most part, **Regular Expressions** concentrates on general principles or concepts that are likely to apply to several scripting languages: inter-process communication, character encodings, graphical toolkit bindings, concurrency, and so on. Three and a half years ago, though, we did a feature article, "Choosing a scripting language," when *Unix Insider* was still *SunWorld*. It's time to update that piece.

Wiersba's question is a good one. He wants to make the most of his education and experience, so he's looking for a language that won't limit him. Rather than learn several poorly, he's out to concentrate on one and get to know it well. Which one deserves his time and attention?

The technical capabilities of most of the popular modern scripting languages have converged remarkably, even over the three years that we've been writing **Regular Expressions**. You can largely expect to be able to do the same things with Perl, PHP, Python, Rexx, Scheme, Tcl, and other such languages: script dynamic Webpages, build administrative utilities, prototype graphical user interface applications, glue together legacy data and processes, and so on. You can code in procedural, object-oriented, or functional styles. You can expect to be able to run your scripts on Mac OS, Unix, or Windows. You can manage other processes, and script Java and COM objects. You have limited access to hardware such as serial ports, but you can extend built-in capabilities with C or other languages. Internationalization is possible and often practical. You can freely reuse any of a staggering number of modules other programmers have written.

Therefore, our first answer is to use what feels comfortable to *you*. If Python's use of white space makes your skin crawl, learn Ruby instead. If you want to get along with your coworkers without having to explain your Tcl or Forth mentality, you might find them ready to accept Perl or Pike. You'll be able to create roughly the same functionality no matter which language you use.

Collect your own data, talk to the natives

The successes in language choice are often instances in which a programmer gets along well with the other natives. What's most important about Perl is probably not the syntax of the language or its constellation of command-line arguments, but the cultural system of the Comprehensive Perl Archive Network (CPAN) -- maintained perldocs, good conferences, local PerlMonger meetings, and so on -- that makes practitioners feel they've found a home.

One of the most pleasant aspects common to these languages is that you can take them out for a spin inexpensively. Locating a language processor and documentation, downloading them to a host you control, and writing your first sample program usually fits within a conventional lunch break. There are extensive online information archives available for all these languages. Rather than flip a coin or trust an expert to decide, you can

quickly acquire your own sense of how different languages work.

Each language does have limits. Their analysis, though, has become exceedingly arcane. Advanced use of Python, for example, rests crucially on its aesthetic for introspection and its promotion of metaprogramming. Is that the right approach for you? There's almost no adequate way to judge this without the subtle understanding of you and your work that no one else is likely to have.

Still, we can give you a starting point. If you follow one of our recommendations, and it takes two years to find out that another choice would have been better, we'll feel successful.

Scouting notes

Here's what we think about the leading portable scripting languages:

Perl: Perl outranks all the others quantitatively. More job seekers claim Perl experience, and more companies advertise its use, than all the others combined. Perl is powerful, but *not* easy. Perl is ubiquitous in Web work and Unix system administration. Greatest liability: extending Perl with C is tedious, and with Java an adventure.

Python: Python's grown rapidly over the last couple of years. Python has no particular weaknesses: It's a great language for beginners, it's a great language for large projects and teamwork, and it enjoys a breathtaking array of convenient accessories. Python is the one language with which we feel safe in recommending for all situations. It's the market leader in several areas outside the United States. It's a good, clean language, and many important applications have already been written with Python. Greatest liability: Tkinter is slow for some applications.

REBOL: REBOL gives good multipliers. Inexperienced programmers do powerful things with it quickly, but it has enough substance to be satisfying for professionals. If you specialize in the kinds of network-aware applications REBOL now seems to be targeting (e-communications and e-commerce), consider this language. Greatest liability: it's a proprietary product. This is also an asset, in some situations, of course.

Ruby: Ruby's growth rate in the last year has been even greater than Python's, although from a smaller base. Like Python, it's won over many former Perl programmers looking for stronger object orientation or a cleaner language. Ruby is tops in Japan, where it was invented. Greatest liability: there's a lot left to write, including GUI bindings, Web modules, packagers, and many specialized libraries.

Tcl: It's a mystery to us and to several companies in the business why Tcl's buzz has quieted. Book sales have fallen off dramatically, and new programmers show little interest in it. Tcl and its extensions are indispensable in numerous areas of computing, though, and the languages will be actively used and developed for a long time to come. Tcl is a bit of an outlier syntactically: while those outside computing often take to it, as a language it doesn't feel nearly as close to C and Java as do Perl and even Python. One of the benefits of this strangeness is that piping and TCP-level networking are much easier with Tcl than with the other languages. Greatest liabilities: inadequately documented Win* interfaces, including COM, and faltering Mac OS support.

We have a lot of affection for plenty of other languages, including Forth, Smalltalk, Scheme, PHP, and Eiffel. The ones above, though, dominate our correspondence. For more details, see the references in the resources below. We particularly welcome reports of your own experiences in the Scripting Languages & Techniques forum we moderate.

Unix Insider