Why perl is my favorite language (or "what makes a good programming language?")

Rolfe Bozier

20-Aug-2014

Agenda

- Background
- What makes a good language
  - Objective aspects
  - Subjective aspects
- Why perl?
- Take-home message

Background

- Some questions this talk may help you answer…
  - Why is a good language important?
  - What is a good language?
  - What language should you choose next time you have a choice?
  - What language should you learn next?
  - How could you become more productive when you write a program (at work or at home)?

Personal agendas and biases

- Criteria for evaluating a language can be subjective or objective
  - “subjective” = your/my opinion
  - “objective” = someone else’s opinion
- My personal experience:
  - I've been paid to write/maintain software in C, Perl, Python, C++, Java, Fortran, Javascript
  - also various DSLs: SQL, LaTeX, PostScript, PDF, sendmail, nroff
  - At home I’ve also written a reasonable amount of PHP and Verilog
  - I've read books on OCaml, Haskell and Ruby :-)
- I don't claim to be a language expert, but I'm interested in them
  - I’ve given up looking for the silver bullet
- My biases are towards open, freely available tools
  - I've got no experience with MS languages: VB, VC++, C#, F# etc.
  - I've barely done any GUI-related work

Objective assessments of a language

- Does it follow the “right” paradigm: imperative, declarative, functional, OO, logic, symbolic, ...
  - http://en.wikipedia.org/wiki/Programming_paradigm lists many more
  - Really, a lot of them are somewhat domain specific, but that doesn't stop proponents from advocating blanket adoption of their choice
- Some claims about what makes a language better:
  - OO is better than imperative (encapsulate everything into an object)
    - *but everything *isn't* an object*
  - Functional is better than both (no side-effects)
    - *programs without side-effects are not useful*
  - Easy to learn
    - *but you only need to learn a language once*
  - You should be constrained in how you write ("only one way")
    - *but what if there is then *no* way to solve your problem?*
  - It should be compiled / interpreted / executed in a VM
  - Should be clean, easy to read, minimal syntax, dense concepts
  - Etc. etc.

Objective assessments of a language

- Who decides these?
  - academics, researchers, language authors.
- Who measures this and how?
  - ???
- How much has software quality improved in the last 30 years?
  - Which languages have made significant improvements to software?
  - How much is software quality really down to
    - individual programmer,
    - environment and culture,
    - tools,
    - development processes,
    - available time and money,
    - Etc.
  - The Space Shuttle application software was about the size of XEBRA, but had about 3 bugs
    - should we adopt HAL/S (a dialect of PL/I)?

Subjective assessments of a language

"The best language is the one that gets the job done in the least amount of effort and time"

- Maybe *you* are in the best position to evaluate the language you should be using

Subjective assessments of a language

- "… gets the job done in the least amount of effort and time"
  - What is the problem you need to solve?
- You have a problem/solution space, which could include:
  - the business area you are in
  - a data repository you need to analyse
  - systems you need to interface with
- You have constraints:
  - execution time
  - size
  - environment
  - resources
  - time, money, staff
  - or maybe you have no constraints (you are your own customer)
- The job you need to do today may be different next week, next year or on the weekend

Subjective assessments of a language

- "… gets the job done in the least amount of effort and time"
  - What does your solution need to do?
- Completion of the job includes:
  - solving the problem (duh!)
  - …until you need to revisit it
    - maintainability
    - extendibility
  - ensuring that it is reliable
  - meeting performance and scalability goals
  - other non-functional requirements of varying importance

Subjective assessments of a language

- "… gets the job done in the least amount of effort and time"
  - How hard is it to implement the solution?
- The language should support a solution to your problem
  - don't bother trying to write a web server in Postscript
- The language should provide the tools to add functionality as efficiently as possible
  - no need to re-invent the wheel again!
  - built-in abstractions, modules, libraries, frameworks, ...
  - who wants to write another hash table implementation?
- As well as making code easier to write, they also make it easier to read
- A language is much more than just the grammar and a compiler

Subjective assessments of a language

- "… gets the job done in the least amount of effort and time"
  - Your time is important
- Your language should support efficient development
  - short development/test cycles
  - good debugging / diagnostics
  - built in static analysis
  - good development tools, if necessary

Perl features

- Why I like Perl…
- Strengths: text manipulation, regular expressions, interfacing with other systems, concise code
- Multiple paradigms: imperative, object-oriented, functional
- Written by a linguist:
  - Many small built-in functions and operators
  - “More than one way to do it”
  - “Easy things should be easy, hard things should be possible”
- Scalars, arrays, hashes, functions, references
  - Run-time type checking
  - Reference-counting memory management
- On-the-fly bytecode compilation and execution
- Comprehensive local documentation

What others say

- It is interpreted, so it is slow
  - It's not that slow, and in most cases it is fast enough
- Perl is line noise / unmaintainable
  - Most languages are unreadable if you aren't familiar with them
- It is too easy to write bad perl
  - You can write bad code in any language
  - www.ioccc.org
- It is not object-oriented
  - You can use classes and methods if you want
- The learning curve is steep
  - I'd argue there are harder ones out there
  - You don't have to use all the language
- It is write-only language
  - Whatever that means

Interesting features

- CPAN – a repository of 137,000 packages
  - Installation via a single command, including running the test suite
- Some interesting language features
  - Classes and objects, if you want them
  - Default variable: $_
  - Arrays and hashes are first class data types – concise syntax makes them very efficient to use
  - Tainted variables – contain user-supplied data
  - Closures – functions with associated stack data
  - Tied hashes – act like a hash, but you define the operational semantics via callbacks

A few modules I have used

- Some random modules I have used:
  - LWP::Agent – turn a script into a web client with a few lines
  - MP3::Info – query and update the information tags in MP3 files
  - Mail::MboxParser – search and extract information from mail folders
  - DBI – access to various databases via standard API
  - HTML::LinkExtor – get the links from a web page, great for writing a spider
  - CGI – create and parse web forms
  - IO::Uncompress::Inflate – decompress some data in a firmware blob
  - ClearQuest – you can guess what this talks to…
- Each one embeds powerful functionality with very little code – all the hard work is done

A simple example

```
#!/usr/bin/env perl
use strict;
use warnings;

my %dict = ();                              # create a hash to store the words

while (my $line = <>)                       # read a line from a file on the cmd line
{
    foreach my $word (split /\s+/, $line)   # split the line into whitespace-separated words
    {
        $dict{$word} = 1;                   # add each word to the hash
    }
}

#
# sort and print the words that start with “r”, in a functional style
#

print                                       # 5. print the resultant array
    sort                                    # 4. sort the words lexicographically
      map { "$_\n" }                        # 3. for each word, append a newline
        grep { m/^r/ }                      # 2. filter out the words starting with “r”
          keys %words;                      # 1. get the hash key values as an array

exit 0;
```