I wish Ruby and Python stopped changing so much

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:47:22Z

In my article on Go, I expressed my frustration on trying to make *gdrive* work in no time. Unfortunately, I failed to do so in a predicted time back then. Many users commented on wrong title and problems with the content. While I still believe tools that just don't work are likely to drive users away, I must say that one of the reader's comments on my article expressed a point of view, which I can identify with:

> Please don't event try to convince me to rubygems/bundler which is a pain in the ack, regardless if I'm using rbenv, RVM or similar helpers... not to mention about npm for nodejs... same with python virtualenv and python2/3 incompatibilities...

Even though I use RVM on my MacBook Air and recently, it's been pretty flawless, I admit that RVM mixed with Rails can sometimes be like to an explosive mix. Especially, when one tries an older version of Rails on a new system, or vice versa.

I believe everyone who writes software understands that some early design decisions are very important. It's hard to predict how APIs will evolve over time and what the final use cases will be like. In other words, we all need to fix programs, upgrade, and move on. In a self-contained world, it's pretty easy. However, for things, such as operating systems and programming languages, it's very hard because once deployed, the features and bugs are there to stay.

Keep in mind, there are also some little things, which can cause some problems for users, such as:

macb:24:10: W: File.exists? is deprecated in favor of File.exist?.

if File.exists?("obj") then

^^^^^^^

macb:47:11: W: File.exists? is deprecated in favor of File.exist?.

if !File.exists?(“makefile”)

^^^^^^^

macb:74:10: W: File.exists? is deprecated in favor of File.exist?.

if File.exists?(obj_fn) then

^^^^^^^

It’s a fairly minor change. The change from “print expression” to “print(expression)” in Python is much harder, since it potentially breaks programs as complex as “Hello world”. In terms of the cited example, I’m not in favor in any of the method names, but once it was picked, I think it should stay. You may ask, “Why?” Well, probably someone somewhere in their scripts already decided to use “File.exists”, which is quite a common function. And, maybe they wanted to leave their scripts working fine after the system upgrade.

There are posts around which mention the same problem but I wonder if other users are experiencing these issues as well. Python 2 to 3 upgrade is no different, and, based on the length of the list, it makes me feel that, for deployment purposes, the two versions may need to be treated as two separate languages.

In my experience with technologies, only C can be pulled from a 1980 `tar.gz` file, and the program is likely to compile. We’re talking about standard C programs with clean, standard `Makefile` files. Somewhere in 1990 the `autotools` suite started to be popular, since UNIX was so fragmentated that programmers didn’t know whether to use `#include <string.h>` or `#include <strings.h>` and what to include to use UNIX domain socket.

From the languages which I’ve used, only Perl has tried to address the problem. What do I mean by that? Well - go ahead and open the script in old Ruby or fairly old Ruby on Rails application. Look at both versions. Would you be able to tell which is which? Same for Python 2 vs Python 3 script. Without actually trying to compile it, it’s hard to tell whether the script is the right one of the interpreter version which one has. Perl solution has its problems but would be one of the alternatives.

Another idea would be more radical: how about if we made small libraries define *syntax* of the language. The microkernel of the language would never change and have around 20–30 primitives which would never change (Forth and Lua are like that). And to get `print` method, you’d have to include the library. But this leads us with the hell of includes and delivering very old version of basic primitives. Also this doesn’t solve the problem, but pushes it somewhere else. With this approach, we’d have to test the whole library of primitives and ensure it still works.

This situation makes me wonder out loud, “What are the best patterns to keep the language evolving without breaking backward compatibility?”