Programming Languages (CS302 2007S)

Rebelsky - Scripting the GIMP with Script Fu

Comments on:

Rebelsky, Samuel (2007). Scripting the GIMP with Script-Fu. Web resource available at `http://www.cs.grinnell.edu/~rebelsky/Courses/CS302/2007S/HOG/script-fu.html` (last modified 30 January 2007; visited 30 January 2007).

Comments

The use of LISP as a tool to aid users of the GIMP seems to blur the line between user interface and programming language. LISP is itself a programming language, but in this case it is being used as a means to provide the user with a powerful, generalized way to automate various tasks within the GIMP. It seems like there are some parallels that could be drawn between "good" programming language design and "good" user interface design.

Questions

Doesn't Photoshop have scripting support for JavaScript and some other languages? *You may be able to script it a bit in Applescript on the Mac. However, I don't think it is scriptable to the same level that the GIMP is.*

I can see the advantages for script support especially in animation, and when people with no artistic talent like myself want to create an image, but is there a definite advantage for skilled artists creating a single image? Are there some processes that are much easier to script then to draw by hand because I feel like a skilled artist could create an image much faster by hand than with a script or a script's aid?

I expect that most artists would want to use something like ScriptFu either (a) to manipulate an exiting image (e.g., to create a filter) or (b) to explore a design space. This is, however, a question that I hope to explore over the next few years.

I realize that there are times when its useful to enter commands by typing them, but in terms of languages being more user friendly wouldn't graphical user interaction be more popular and easier? What are some of the other advantages to textual computing besides the one mentioned in the reading?

The GIMP provides both a graphical UI and a scripting console. As the comment above suggests, sometimes providing text helps the UI. We'll explore these questions a bit this week.

Has anyone designed a kind of hybrid between the terminal and the graphical types of interfaces? Like basically a point and click that allows you to specify/asks things of you when you click. Or if you wanted to, as the example is given, click a particular point on the screen, having a window over your mouse tracking it's location, or having specific keys that you could invoke to move it in increments?

Well, the Ryder reading did mention visual languages, which can do some of those things. Unfortunately, UI languages are not my area of specilization.

What is the differentiating factor between Scheme and Script-Fu that warrants Script-Fu to be called a language? From looking at the sample Script-Fu code, it seems as though Script-Fu is just Scheme with built-in libraries for image manipulation. If this is the case, I would think that Script-Fu is just a dialect of Scheme.

Yup, just a dialect. Dialects of languages are still languages, though.

What other scripting languages does GIMP support?

The first version supported only Script-Fu. Now, it also supports, um, Python, I think. I'm not sure whether there's a built-in console though.

Is Script-Fu Object-Oriented?

Only insomuch as Scheme is object-oriented. We do build object-like things in Scheme that conceal their fields, as you should remember from 151 or 153. However, there is no built-in inheritance. (Polymorphism is a bit harder; you can certainly send any object any message, but it's rare that this will give you much benefit.)

Pedagogy

The paper states a good point that using scripting for image is handy. However, it doesn't convice me why you have chosen Script-Fu.

I haven't chosen Script-Fu; the designers of the GIMP did. If you're wondering why I chose it for this class, it's because I want to see whether the labs and readings help you better understand higher-order programming.

I thought the reading for tomorrow is quite straight-forward. I know that you tried to tie this in to your CS151 classes last year. Did you find that having a visual results helped students understand functional programing? What were your other reasons for incorporating Script-Fu to your classes?

The question is still out as to whether the graphics helped students understand some key higher-order concepts. I think it helped with sectioning. I also wanted to find something that would be fun, that would show a different kind of programming, and that showed a "real world" use of Scheme.

Why Scheme?

A lot of questions are about the choice to use Scheme (and SIOD) in the GIMP.

What's the strength of Script-Fu?

Also, why did the GIMP creators consider Scheme (particuarly the functional approach) a good scripting language? And, once they chose scheme, why did they choose SIOD? From your library, it seems like its missing many elementary functions and predicates (e.g. - even? integer? list->vector etc) are seemingly very useful for images?

One of the things I would like to pick up from this class is how to decide on what programming language to use when working on a project. What are some of the reasons the developers of Script-Fu chose to make it a "scheme-like" language? What are the advantages of making Script-Fu "scheme-like" as opposed to making it similar to any other widely used language?

I'm not sure they've written a lot about the decision. My understanding is that it's mostly that Scheme is small, and widely appreciated in the open-source community. (Okay, FSF pushes Scheme; I'm not sure about the rest of the open-source community.) A nice, portable, implementation that is easy to extend (SIOD) was also available.

There are certainly flaws to SIOD, one of which is that it implements a relatively small subset of Scheme. My guess is that the decision was primarily one of convenience.