10 ways GNU Guile is 10x better

#4 will shock you! :-)

In Rust at Facebook by Fitzhardinge of the Mercurial team, Jeremy says that a new language must be 10x better at **something** than one of the (other) incumbent languages.

So I asked on IRC: what's the 10x advantage of **Guile**?

This is not „which Scheme to choose“ (aside from one specific feature). For that question, see the opinionated guide to scheme implementations. For Guile's 10x advantages (some due to being Scheme), read on.

Contents

1 powerful core 2
2 runtime introspection and modification 2
3 s-expressions and homoiconicity 3
4 interfacing with C and access from C 6
5 fibers 6
6 embedded natural script writing 7
7 hackability 8
8 complete info-manual 9
9 prototyping and creativity 9
10 lots of fun 9
11 More 10x advantages 10
List of Links 10

1 powerful core

Macros (`define-syntax`) and *delimited continuations* enable creation of high level concepts like fibers without having to change the core.

Thanks to the efficient compiler, there is rarely a need to use macros for performance instead of for semantics, so your code stays cleaner.

thanks to stis.

> "Delimited continuations are superior to Python's yield, the macrology is superior to any language except Scheme, the hackability is better then essentially all closed sorurce solutions of program languages.
> ...
> And the efficient inlining of lambdas is perhaps matched by C, but not many higher languages. The design of Scheme really shines here compared to e.g. Python." — stis who is building Python on Guile

2 runtime introspection and modification

Compared to C and Go, runtime access to running code is very useful. You can jump into a module and modify anything during runtime.

thanks to str1ngs for this point.

While you even get a subset of this with Java using incremental compilation and hot reloading of code in IntelliJ, at work we're always dancing around changes that change some method arguments or streams, because those break hot reloading so you have to restart.

And for Javascript we can in theory replace every function, but in practice at work we transpile everything with babel and webpack and that breaks hot reloading, so we have to reload after every change.

In Guile you can either start in a REPL, or create a dedicated REPL in the running program, or create a REPL socket for your development environment and connect to that to hack on the running server.

Then you can re-define all top-level definitions in all modules.

As example str1ngs usually develops the Nomad web browser like that.

And if you need a game loop as main thread, you can use it to drive the cooperative repl server.

3 s-expressions and homoiconicity

Compared to non-lisp languages, the regularity of s-expressions and being able to treat code as data and the other way round (homoiconicity) is a big advantage.

thanks to pinoaffe for this point.

This is an essential elegance I want to conserve in wisp.

Wikipedia notes as advantage that

> extending the language with new concepts typically becomes simpler, as data representing code can be passed between the meta and base layer of the program. — Homoiconicity: Uses and Advantages

and

> It can be much easier to understand how to manipulate the code since it can be more easily understood as simple data (since the format of the language itself is a data format). — Homoiconicity: Uses and Advantages

For Wisp I use this a lot, because it allows me to do a first simple pass over the code, add incremental improvements and finally have the cleaned up code that I can pass to the language spec definition:

```
define : wisp-scheme-read-chunk port
         . "Read and parse one chunk of wisp-code"
     let : :  lines : wisp-scheme-read-chunk-lines port
          wisp-make-improper
            wisp-replace-empty-eof
              wisp-unescape-underscore-and-colon
                wisp-replace-paren-quotation-repr
                  wisp-propagate-source-properties
                    wisp-scheme-indentation-to-parens
                     ↪  lines
```

Also this enables me to write a Question-Asking macro for dryads wake without going totally insane. Usage:

```
Choose
    : new game
      ,(first-encounter)
    : load game
      ,(load-game)
    : show prologue
      ,(prologue) ,(welcome-screen)
    : exit
      We hope you enjoyed our game!
```

Definition:

```
define-syntax-rule : Choose . choices
   . "Ask questions, apply consequences"
  begin
    say-lines : ("") ;; newline before question
    let loop :
      ;; Get the response via the Ask macro
      ;; to evaluate inline-code incrementally
      define resp : string->number : Ask choices
```

```
      or
        cond
          : equal? resp 1
            Respond1 choices
          : equal? resp 2
            Respond2 choices
          : equal? resp 3
            Respond3 choices
          : equal? resp 4
            Respond4 choices
          : equal? resp 5
            Respond5 choices
          : equal? resp 6
            Respond6 choices
          else
            . #f
        loop

define-syntax Ask
  lambda (x)
    syntax-case x ()
      : _ (choices ...)
        #` begin
           ask (QuoteFirsts (choices ...))
```

QuoteFirsts interprets the questions as data but leaves the answers as code which can e.g. use `,(load-game)` to open the load game dialog.

4 interfacing with C and access from C

Most of Guile procedures can be called from C (for example when embedding Guile) and it is easy to interoperate with C from Guile Scheme.

thanks to str1ngs for this points, too.

I did not embed Guile in a C-program myself yet, but Guile provides detailed examples and tutorials for this in its Documentation, with C interfaces explicitly documented for most of its procedures.

*This is the **shocking** number 4: Guile is better for writing C-Programs than C itself. Consider yourself **shocked** :-) — and no, this argumentation is not complete. But if you're here, the title led you here. #legitbait.*

*If you want to be **seriously shocked**, look at c-indent. Yes, that is Guile.*

5 fibers

Fibers provide lightweight threads in Guile without having to change anything in the core of Guile. They are reasonably performant and provide concurrency as with Go-channels.

thanks to stis for this point.

With the name giving `fibers` and `channels` they provide an efficient and scalable model for coordinated concurrency. See the manual for details.

You can test their raw efficiency using either the webserver-benchmark or the guile-fibers entry in the skynet benchmark.

> fibers is better than kludges like marking procedures and what not and it is matched by very few languages. — stis

6 embedded natural script writing

For me, one 10x advantage over every other language is that I could integrate wisp and get an embedded script writing language for dryads wake (as embedded domain specific language: eDSL). There's a talk that compares this to other approaches (including ones I tried before): Natural script writing with Guile. Here's a real example:

```
define : first-encounter
    Enter : Juli Fin :profile juli
            Melter Lark :profile melter
            Rooted Breeze :profile dryad
            Old One

    Print
        Please choose your name
    game-state-init!
    game-state-name-set! : read-line
    game-state-id-set! : name->id : game-state-name
    game-state-scene-set! first-encounter
    save-state : game-state-id
    Print
        Welcome ,(string-append (game-state-name) "!")

    Juli Fin
        Finally we have our own home!

    Melter Lark
        It took long enough.

    Juli Fin
        And it is moist for sure.

    Melter Lark
        I will dry it out.
```

```
Rooted Breeze :eerie
    My slumber breaks
    my mind awakes
    who are you strangers
    in my home?

Old One
    How do you answer?
    Juli is ,(score->description
    ↪  (profile-ability-score (game-state) 'juli
    ↪  'explain))
        . at explaining
    and ,(score->description (profile-ability-score
    ↪  (game-state) 'juli 'fast-talk))
        . at fast-talk.
Choose
    : explain your situation to appease the dryad
      ,(explain-your-home)
    : fast-talk the dryad to get her to leave and
    ↪  risk her anger
      ,(fast-talk-the-dryad)
```

7 hackability

The language tower and infrastructure make it enjoyable to hack on and extend Guile itself.

thanks to stis for this point.

> language tower and effective syntax extensions (define-syntax, etc.) - without that, lokke might well not exist (such as it is), at least not by now. — rlb (lokke is Clojure on top of Guile)

8 complete info-manual

Guile comes with a complete and very readable manual in info-format. If you've ever tried to program Python without internet access (or just without Google), you'll know to deeply appreciate this.

You can find most answers by running

```
info guile
```

in the shell, or calling

```
C-h i m Guile Reference
```

in Emacs and starting a full-text search with `C-s {search terms} C-s`.

9 prototyping and creativity

Compared to Python and C++ Guile removes barriers to abstraction. This is also possible with R, but Guile provides good enough performance for these abstractions to make them practical to use.

The low startup time helps for this, too.

thanks to vijaymarupudi for this point.

> "Guile is particularly great for prototyping and creativity, and its performance allows for such experiments to be deployed to the world with minimal changes.
>
> I've been using it a lot lately for data cleaning and modeling, and it's been great" — vijaymarupudi

10 lots of fun

Hacking in Guile/Scheme is just lots of fun.

thanks to dsmith for this. Even though this sounds small and gets only a single line in this article, looking at the enthusiasm of people who hack on Guix shows that dsmith clearly has a point!

11 More 10x advantages

After I wrote this article, people noted more advantages. I might work them into the article sometime later, but for now they live here:

- **named lets**, I could be without map reduce and all that, just let me have a named let construct — stis
- **python-on-guile** has copyable generators and yield as an actual function that can be passed to other functions — stis writing about Python on Guile
- Purely Functional Datastructures and fectors.
- Hoot lets you port your tools to the browser with minimal overhead — including games like Goblinville.
- the **minimalism** *"design not by piling feature on top of feature, but by **removing the weaknesses and restrictions** that make additional features appear necessary"* that allows teaching programming from first define to deployment on 64 pocket-sized pages.

List of Links

draketo.de: https://www.draketo.de 1
Rust at Facebook by Fitzhardinge of the Mercurial team: https://www.youtube.com/watch?v=kylqq8pEgRs&list=PL85XCvVPmGQhDOUIZBe6u388GydeACbTt&index=8 1
Guile: http://gnu.org/s/guile 1

opinionated guide to scheme implementations: https://wingolog.org/archives/2013/01/07/an-opinionated-guide-to-scheme-implementations . . . 1
fibers: https://github.com/wingo/fibers . . . 2
Python on Guile: https://www.mail-archive.com/guile-devel%40gnu.org/msg15805.html . . . 2
Nomad web browser: http://www.nongnu.org/nomad/ . . . 3
cooperative repl server: https://www.gnu.org/software/guile/docs/docs-2.2/guile-ref/Cooperative-REPL-Servers.html . . . 3
homoiconicity: https://en.wikipedia.org/wiki/Homoiconicity 3
I want to conserve in wisp: http://www.draketo.de/proj/wisp/why-wisp.html . . . 3
Homoiconicity: Uses and Advantages: https://en.wikipedia.org/w/index.php?title=Homoiconicity&oldid=1037434473#Uses_and_advantages . . . 3
Homoiconicity: Uses and Advantages: https://en.wikipedia.org/w/index.php?title=Homoiconicity&oldid=1037434473#Uses_and_advantages . . . 3
dryads wake: https://hg.sr.ht/~arnebab/dryads-wake/ . . . 4
in its Documentation: https://www.gnu.org/software/guile/learn/ . . . 6
#legitbait: https://www.youtube.com/watch?v=S2xHZPH5Sng . . . 6
c-indent: http://sph.mn/computer/guides/c/c-indent.html . 6
manual: https://github.com/wingo/fibers/wiki/Manual . . 6
webserver-benchmark: https://github.com/wingo/fibers/tree/master/benchmarks . . . 6
guile-fibers entry in the skynet benchmark: https://github.com/atemerev/skynet/blob/master/guile-fibers/skynet.scm . . . 6
wisp: wisp.org . . . 7
dryads wake: https://hg.sr.ht/~arnebab/dryads-wake/ . . . 7
Natural script writing with Guile: https://fosdem.org/2017/schedule/event/naturalscriptwritingguile/ . . . 7

lokke: https://github.com/lokke-org/lokke 8
info-format: https://www.gnu.org/software/texinfo/ 9
Guix: http://guix.gnu.org 10
Python on Guile: https://www.mail-archive.com/guile-devel%40gnu.org/msg15805.html 10
Purely Functional Datastructures: https://github.com/ijp/pfds 10
fectors: https://github.com/ijp/fectors 10
Hoot: https://spritely.institute/hoot/ 10
to the browser: https://spritely.institute/news/building-interactive-web-pages-with-guile-hoot.html 10
including games: https://spritely.institute/news/make-a-game-with-hoot-for-the-lisp-game-jam.html 10
Goblinville: https://spritely.institute/news/goblinville-a-spring-lisp-game-jam-2025-retrospective.html 10
from first define to deployment on 64 pocket-sized pages: programming-scheme.org 10