Cool, I want to get started!

Great to have you on board! Chances are your operating system already packages Chicken Scheme. Try that first. Once you have it installed, you can try some code with the **C**hicken **S**cheme **i**nterpreter *csi*.

```
$ csi

CHICKEN
(c)2008-2010 The Chicken Team
(c)2000-2007 Felix L. Winkelmann
Version 4.6.2
openbsd-unix-gnu-x86 [ manyargs dload ]
compiled 2010-10-18 on deepthought (OpenBSD)

#;1> (print "Welcome to Chicken Scheme!")
Welcome to Chicken Scheme!
#;2> (* 3 (+ 3 4) (/ 4 2))
42
#;3> ,q ; quits
```

If you aren't familiar with Scheme yet, you can check out the extensive list of books at `http://www.schemers.org/`.

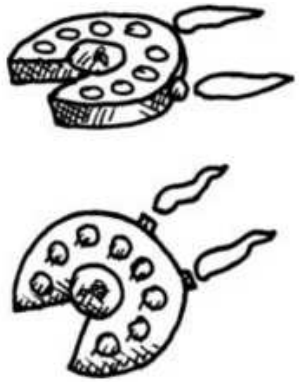
For Chicken Scheme specific documentation, try our wiki. If you want to be kept up to date with developments in chicken you can read a weekly summary called the *Chicken Gazette*.

For Chicken Scheme specific questions there is a newbie-friendly mailing list *chicken-users@nongnu.org*.

The Chicken Team is also reachable on Freenode's IRC, channel #chicken.

Chicken Scheme on the net

Main site:	`http://www.call-cc.org/`
Gazette:	`http://gazette.call-cc.org/`
Code:	`http://code.call-cc.org/`
Chat:	`#chicken at irc.freenode.org`
Documentation:	`http://wiki.call-cc.org/`
About Scheme:	`http://www.schemers.org/`



The Chicken Logo has been made by Joshua Griffith. The other images are drawn by Conrad Barski, used with his kind permission. Also have a look at his book *"Land of Lisp"* on `http://www.landoflisp.com/`

Last updated October 2010.

Chicken Scheme

A practical and portable Scheme system

http://www.call-cc.org/

What is Chicken Scheme?

Chicken is a *robust* and *mature* compiler for the programming language *Scheme*. Chicken has been around for over 10 years now, and has a growing group of dedicated users.

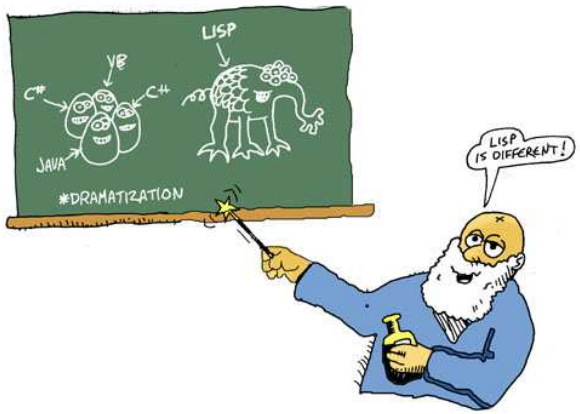
The system compiles to *C*, which allows it to generate *fast, portable code*. For easy development it also includes a powerful interpreter.

Chicken is above all a *practical* system; there are literally *hundreds* of extension libraries (known as "eggs") available to help you get some real work done! There are extensions for web development, concurrency and parallelism, cryptography, scientific computing and much, much more.

All of the major platforms are supported, including GNU/Linux, the BSDs, MacOS X and Windows.

What is Scheme?

Scheme is a truly *elegant and minimal* programming language which directly descends from Lisp, the *second oldest* high level programming language still in use. But don't be fooled by that: It's still as fresh today as when it was first created!



It has very simple syntax so it is *easy to learn*. This also makes it a very powerful tool: writing extensions to the language can be done right from *within* the language, allowing you to *mold the language* to better fit your program's domain. User-added functionality is treated no different from "built-in" functionality.

Scheme is well-known as a *functional* programming language, but it's more accurately described as a *multi-paradigm* language. Sure, you can program in a functional style, but also in all flavors of object-oriented, logical, distributed and even imperative styles. If tomorrow a new cool style of programming is invented, Schemers won't need to switch languages. Instead, they can just add it to their favorite language!

So, why Chicken and not *<insert Scheme here>?*

Scheme is such a minimalist and easy to parse language that there are *thousands* of implementations. However, implementing a *good* Scheme is nontrivial. Of the implementations that are not toys, we prefer Chicken because it:

- sticks to the original minimalist spirit of Scheme
- produces *fast* code
- has a growing number of useful libraries for real-world tasks
- integrates very easily with C
- has been actively maintained for more than 10 years
- has an energetic and enthusiastic community!

What is Chicken Scheme used for?

Scheme is a general purpose language, so your imagination is the limit! Here are just a few examples:

The ease of integrating C libraries and the rapid turnaround offered by the interpreter make Chicken a great prototyping platform.

Many of us automate our sysadmin tasks with Chicken.

It's easy as pie to make web apps with Chicken's web server *Spiffy* and the web framework *Awful*. Our wiki is a good example of this.

You can also generate static web pages using *Hyde*, which powers our weekly *"Gazette"* newsletter.

You can analyze data with our great database support and visualize it with *GNU Octave* or render graphs to *X11*, *SDL* or *OpenGL* windows or *PDF* documents.

This very flyer was produced with SLaTeX, a LaTeX preprocessor which typesets Scheme code blocks with syntax-highlighting. Written in Scheme, of course.

Show me some examples!

Hello world

How boring can it be?

```
(print "Hello, world!")
```

Obligatory factorial program

Slightly less boring than the hello world program:

```
(define fac
  (lambda (n)
    (if (= n 0)
        1
        (* n (fac (- n 1))))))

(define number
  (string->number (car (command-line-arguments))))
(print "The factorial of " number " is " (fac number))
```

Integrating Scheme with C

```
(define log10
  (lambda (x)
    (/ (log x) (log 10))))

;; The same using log10 from C:
(foreign-declare "#include <math.h>")

(define log10-from-c
  (foreign-lambda* double ((double x))
    "double y;"
    "y = log10(x);"
    "C_return(y);"))

;; Shorter versions:
(define (log10 x) (/ (log x) (log 10)))
(define log10-from-c
  (foreign-lambda double "log10" double))
```

As you can see, calling C functions is quite easy. If you have longer C code, you can also write your functions in a separate C file and compile those against your Chicken program. You can just call those functions from Chicken.