

Racket for Everyone (Else)

Morgan Lemmer-Webber

email: mlemmer@wisc.edu

twitter: @mlemweb

fediverse: <https://octodon.social/@mlemweb>

Christopher Lemmer Webber

website: <https://dustycloud.org/>

email: cwebber@dustycloud.org

twitter: @dustyweb

fediverse: <https://octodon.social/@cwebber>

Racket's target audiences?

CS Freshmen
Middle schoolers

Language researchers

Is something missing?

[???

CS Freshmen

Middle schoolers

[???

Language researchers

Is something missing?

Non-CS/math coders
CS Freshmen
Middle schoolers

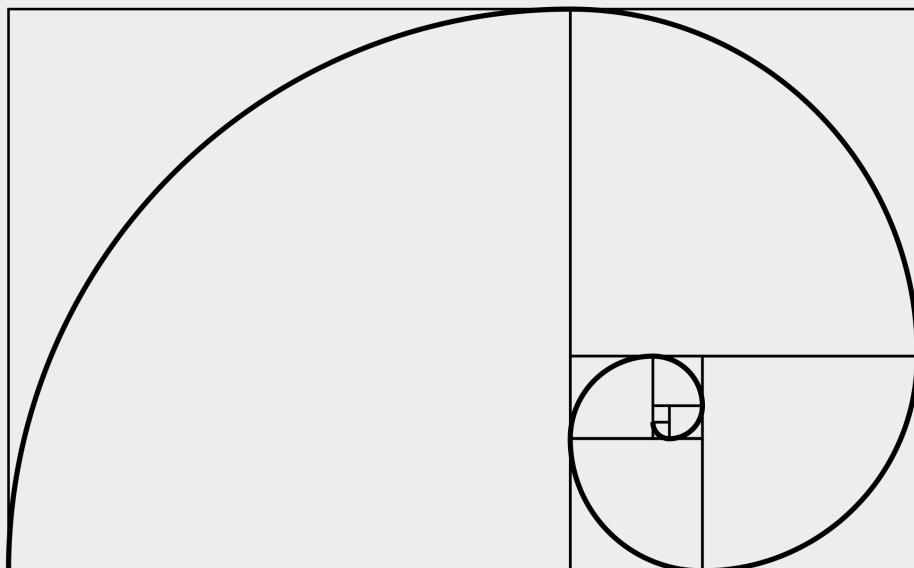
[???

Language researchers

Racket for the Humanities

(or people with CS imposter syndrome)

Learning Python as a non-CS/math person



Programmable Publishing

Digital Humanities for Everyone!



Want to learn computer programming, but don't think you're enough of a math / computer science person? Have we got the workshop for you!

Using the Racket programming language, the user-friendly text editor DrRacket, and the markup language Scribble, participants will learn:

- The basics of learning to program... not by crunching numbers, but by making pictures!
- Use Scribble and DrRacket to write papers in one source file, but export to HTML, PDF, and other formats!
- Integrate data analysis right into your documents! How many times *did* Jane Austen mention tea in her writings?
- Write custom tools to fit *your* research needs. Tired of formatting and reformatting image lists? Why not let your program do it for you?

The workshop will be on **Saturday March 17 from 1-4pm in Elvehjem L170**. This event is sponsored by the UW Madison Material Culture Focus Group and is open to students, faculty, and the community. Please email mlemmer@wisc.edu if you are interested in participating as space is limited.

You *can* learn to program... yes, you!

Pssst... flip this flier up to see its source code!

File Edit View Language Racket Insert Tabs Help

digital-humanities-flier.scrbl (define ...) ▾ Run Stop

★ 1: digital-humanities-flier.scrbl 2: Images.scrbl

```
#lang scribble/base

@(require images/logos
           scribble/sigplan)

@centered{
  @larger{@larger{@larger{@bold{Programmable Publishing}}}}}

@centered{
  @larger{@bold{Digital Humanities for Everyone!}}}

@centered{@(plt-logo #:height 75)}

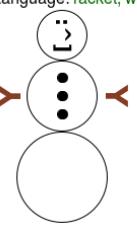
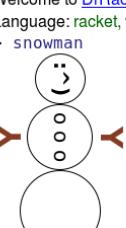
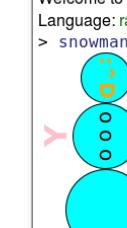
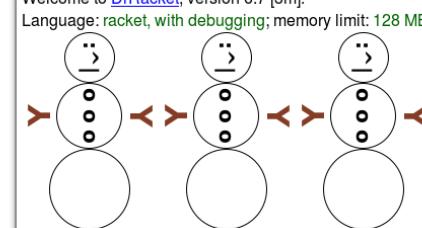
@larger{
  Want to learn computer programming, but don't think you're enough of
  a math / computer science person? Have we got the workshop for you!

  Using the Racket programming language, the user-friendly text editor
  DrRacket, and the markup language Scribble, participants will learn:

  @itemlist[
    @item{
      The basics of learning to program... not by crunching
      numbers, but by making pictures!}
    @item{
      Use Scribble and DrRacket to write papers in one source file,
      but export to HTML, PDF, and other formats!}
    @item{
      Integrate data analysis right into your documents!
      How many times @emph{did} Jane Austen mention tea in her writings?}
    @item{
      Write custom tools to fit @emph{your} research needs.
      Tired of formatting and reformatting image lists?
      Why not let your program do it for you?}]

  The workshop will be on @bold{Saturday March 17} in @bold{Elvehjem L170}.
  This event is sponsored by the UW Madison Material Culture Focus Group.
  Open to students, faculty, and the community. Please email mlemmer@wisc.edu
```

Determine language from source ▾ 4:26 246.57 MB

JessieSnowman.scrbl	MichelleSnowman.rkt	TanyaSnowman.rkt	JesseSnowman.rkt - DrRacket
<p>File Edit View Language Racket Insert Tabs Help</p> <p>JessieSnowman.scrbl ▾ (define ...)</p> <pre>#lang racket (require pict) (define (snowball size) (disk size #:color "white")) (define head (cc-superimpose (snowball 50) (text ":^)" '(bold) 20 (* pi -.5))) (define left-arm (colorize (text "Y" '(bold) 30 (* pi .5)) "brown")) (define right-arm (colorize (text "Y" '(bold) 30 (* pi -.5)) "brown")) (define button (disk 10 #:color "black")) (define buttons (vc-append 8 button button button)) (define body (hc-append</pre>	<p>File Edit View Language Racket Insert Tabs Help</p> <p>MichelleSnowman.rkt ▾ (define ...)</p> <pre>#lang racket (define (snowball size) (disk size #:color "white")) (define head (cc-superimpose (snowball 50) (text ":^)" '(bold) 20 (* pi -.5))) (define left-arm (colorize (text "Y" '(bold) 30 (* pi .5)) "brown")) (define right-arm (colorize (text "Y" '(bold) 30 (* pi -.5)) "brown")) (define butt (snowball 80)) (define snowman (vc-append head body butt))</pre>	<p>File Edit View Language Racket Insert Tabs Help</p> <p>TanyaSnowman.rkt ▾ (define ...)</p> <pre>#lang racket (define (snowball size) (disk size #:color "cyan")) ;; Snowman components (define head (cc-superimpose (snowball 50) (colorize (text ":^D" '(bold) 20 (* pi -.5)) "orange")) (define left-arm (colorize (text "Y" '(bold) 30 (* pi .5)) "pink")) (define right-arm (colorize (text "Y" '(bold) 30 (* pi -.5)) "pink")) (define body (hc-append left-arm (cc-superimpose (snowball 65) (text "o o o" '() 20 (* pi -.5))) right-arm)) (define butt (snowball 65))</pre>	<p>File Edit View Language Racket Insert Tabs Help</p> <p>JesseSnowman.rkt ▾ (define ...)</p> <p>Check Syntax</p> <pre>#lang racket (require pict) (define (snowball size) (disk size #:color "white")) (define head (cc-superimpose (snowball 50) (text ":^)" '(bold) 20 (* pi -.5))) (define left-arm (colorize (text "Y" '(bold) 30 (* pi .5)) "brown")) (define right-arm (colorize (text "Y" '(bold) 30 (* pi -.5)) "brown")) (define body (cc-superimpose (snowball 65) (text "o o o" '() 20 (* pi -.5))) (define body-with-arms (hc-append left-arm body right-arm))</pre>
<p>Language: racket, with debugging; memory limit: 128 MB.</p> 	<p>Welcome to DrRacket, version 6.7 [3m].</p> <p>Language: racket, with debugging; memory limit: 128 MB.</p> <pre>> snowman</pre> 	<p>Welcome to DrRacket, version 6.7 [3m].</p> <p>Language: racket, with debugging; memory limit: 128 MB.</p> <pre>> snowman</pre> 	<p>Welcome to DrRacket, version 6.7 [3m].</p> <p>Language: racket, with debugging; memory limit: 128 MB.</p> <pre>></pre> 
Determine language from source ▾	Determine language from source ▾	Determine language from source ▾	Determine language from source ▾
			4:2 474.83 MB

How To Use Scribble to Write your Academic Papers: A Reference Tutorial

Morgan Lemmer-Webber

March 25, 2018

Abstract

This tutorial gives examples of the common functions and formatting tags you will need to write an academic paper using Dr. Racket and scribble. This document should be used as a reference guide when you are ready to set up your own academic paper.

This document itself is written in Scribble, and will be most useful to you if you look at both the scribble files (HowTo.scrbl, Basics.scrbl, DocumentStructure.scrbl, Lists.scrbl, Tags.scrbl, Links.scrbl, Tables.scrbl, Citations.scrbl, Images.scrbl, Export.scrbl) and the pdf (HowTo.pdf) or html output (HowTo.html) at the same time. You can find these documents on our git repository <https://github.com/mlemmer/DigitalHumanities>.

Both Racket and Scribble have wonderful documentation and tutorials available, see below for useful references. Often times if you are stuck and can't figure out how to write a function or debug your code on your own you can find an answer by simply googling it (be sure to include some combination of Racket/Scribble in your search because there are a lot of programming languages out there). Racket also has a thriving community of programmers, so you can post a question on the [Racket Users forum](#)

Introduction to Racket with Pictures: <https://docs.racket-lang.org/quick/index.html>

Web Applications in Racket: <https://docs.racket-lang.org/continue/index.html>

Racket Guide: <https://docs.racket-lang.org/guide/index.html>

Racket Reference: <https://docs.racket-lang.org/reference/index.html>

Racket Documentation: <https://docs.racket-lang.org/continue/index.html>

Scribble tutorial: <https://docs.racket-lang.org/scribble/getting-started.html>

Scribble Manual: <https://docs.racket-lang.org/scribble/>

1 License

The documentation and content of this tutorial are licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)

All code for this project is licensed under the GNU Lesser General Public License version 3.0 (LGPL) or (at your option) any later version of the GNU LGPL as published by the Free Software Foundation.

2 Basics of the Scribble/Racket language

The first line of your document should indicate the language you are using `#lang scribble/doc`. There should be nothing else on that line. The next element on your document should be any libraries you will

Remember this?

Non-CS/math coders
CS Freshmen
Middle schoolers

[???

Language researchers

Racket for the “General” Programmer

Non-CS/math coders

CS Freshmen

Middle schoolers

General coders

Language researchers



Racket solve problems · make languages

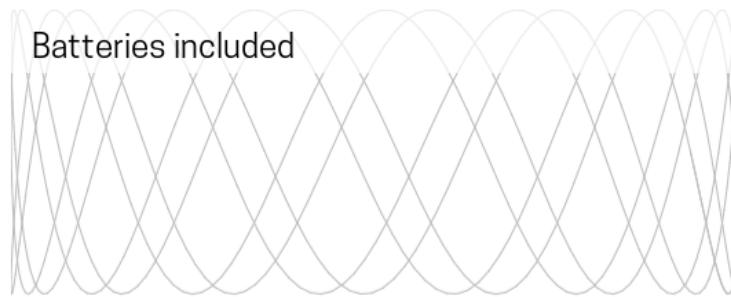
DOCS

PACKAGES

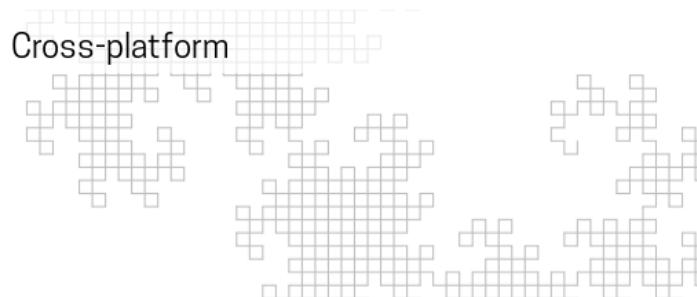
DOWNLOAD

(eighth RacketCon) is for developers, contributors, programmers, educators, and bystanders. September 29–30 in St. Louis, Missouri. [Join us!](#)

Batteries included



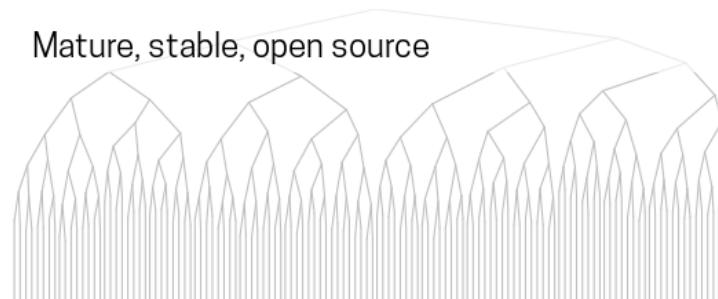
Cross-platform



Powerful macros & languages



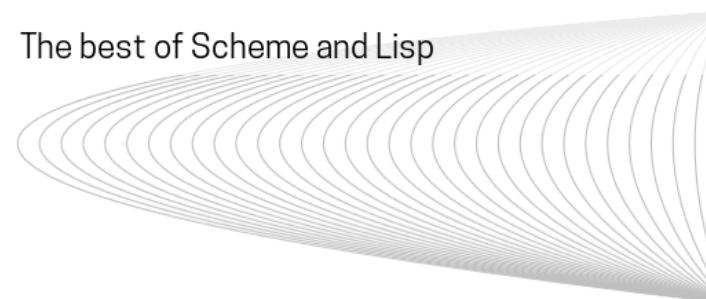
Mature, stable, open source



DrRacket IDE & tons of documentation



The best of Scheme and Lisp



Racket is a **general-purpose programming language** as well as the **world's first ecosystem** for language-oriented programming. Make your **dream language**, or use one of the dozens **already available**, including these —

#lang racket

```
(require 2htdp/image) ; draw a picture
(let sierpinski ([n 8])
  (cond
    [(zero? n) (triangle 2 'solid 'red)]
    [else (define t (sierpinski (- n 1)))]))
```

#lang typed/racket

```
; Using higher-order occurrence typing
(define-type SrN (U String Number))
(: tog ((Listof SrN) -> String))
(define (tog l)
  (apply string-append (filter string? l))))
```

#lang racket/gui

```
(define f (new frame% [label "Guess"]))
(define n (random 5)) (send f show #t)
(define ((check i) btn evt)
  (message-box "." (if (= i n) "Yes" "No")))
(for ([i (in-range 5)])
```

#lang scribble/base

```
@; Generate a PDF or HTML document
@title{Bottles: @italic{Abridged}}
@apply
  itemlist
  (for/list ([n (in-range 100 0 -1)])
```

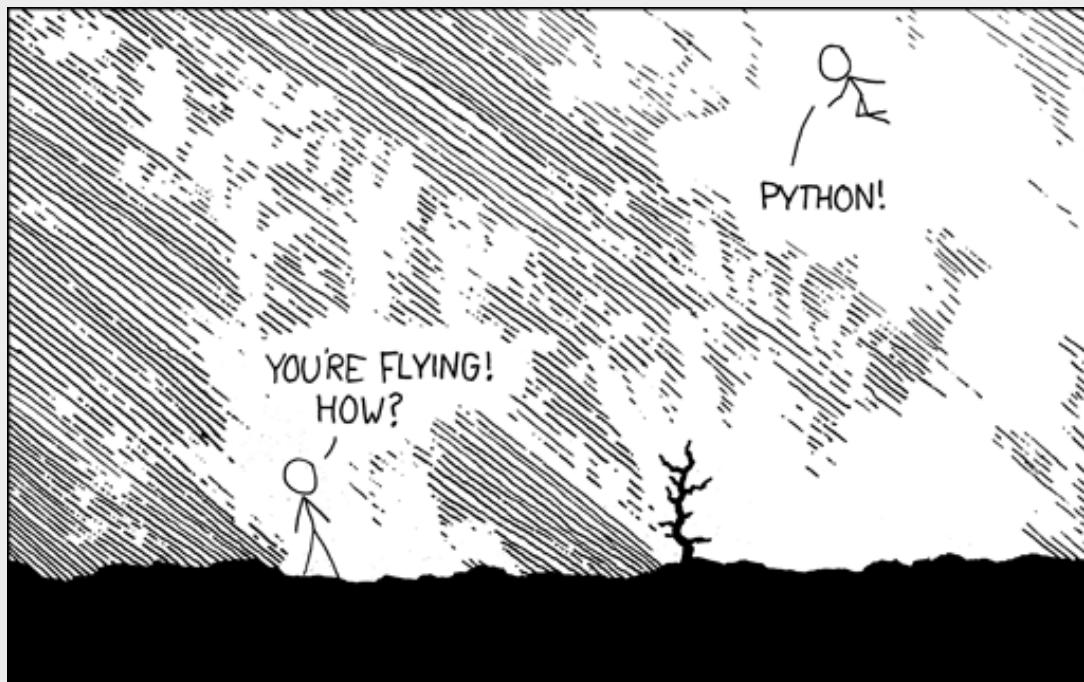
#lang datalog

```
ancestor(A, B) :- parent(A, B).
ancestor(A, B) :-
  parent(A, C), ancestor(C, B).
parent(john, douglas).
parent(bob, john).
```

#lang web-server/insta

```
; A "hello world" web server
(define (start request)
  (response/xexpr
    '(html
      (head (title "Racket"))))
```

This was me for about 10 years...







#langs are great, but libraries are legos

And Racket has so many great legos!

- pict
- web server
- db
- crypto
- game libraries (big-bang, lux, ...)
- ... and so much more!

`(how do i learn ,some-lisp?)

The screenshot shows an Emacs window with two buffers open:

- rainbow-identifiers.el (210,33) [Emacs-Lisp]**: This buffer contains the source code for the `rainbow-identifiers` minor mode. It includes functions for testing face overriding, matching symbols, and toggling the mode. It also defines a minor mode and handles font locking.
- core.scm (138, 0) [Scheme]**: This buffer contains Scheme code for task management. It includes definitions for `let*`, `define`, `run-tasks-until-failure`, `define-syntax-rule`, and `define-task`.

```
(File Edit Options Buffers Tools Emacs-Lisp YASnippet Help)
"Test if the face of the identifier under BEGIN is overridable."
(let ((face (get-text-property begin 'face)))
  (cond
    ((null face)
     t)
    ((listp face)
     (catch 'rainbow-identifiers--face-overridable
       (dolist (face* face)
         (unless (memq face* rainbow-identifiers-faces-to-override)
           (throw 'rainbow-identifiers--face-overridable nil))))
     t))
    (t
     (memq face rainbow-identifiers-faces-to-override)))))

(defvar rainbow-identifiers--face nil)

(defun rainbow-identifiers--matcher (end)
  "The matcher function to be used by font lock mode."
  (catch 'rainbow-identifiers--matcher
    (while (re-search-forward (rx symbol-start (*? any) symbol-end) end t)
      (let ((beginning (match-beginning 0))
            (end (match-end 0)))
        (when (run-hook-with-args-until-failure 'rainbow-identifiers-filter-functions beginning end)
          (let* (((identifier (buffer-substring-no-properties beginning end))
                  (hash (rainbow-identifiers--hash-function identifier)))
                 (setq rainbow-identifiers--face (funcall rainbow-identifiers-choose-face-function has)))
            (throw 'rainbow-identifiers--matcher t))))))
  nil))

;;;###autoload
(define-minor-mode rainbow-identifiers-mode
  "Highlight identifiers according to their names.

Toggle Rainbow Identifiers mode on or off.

With a prefix argument ARG, enable Rainbow Identifiers mode if ARG is
positive, and disable it otherwise. If called from Lisp, enable the
mode if ARG is omitted or nil, and toggle it if ARG is 'toggle'."'
  :init-value nil
  :lighter ""
  :keymap nil
  (let ((keywords '((rainbow-identifiers--matcher 0 rainbow-identifiers--face prepend))))
    (font-lock-remove-keywords nil keywords)
    (when rainbow-identifiers-mode
      (font-lock-add-keywords nil keywords 'append)))
  ; Refresh font locking.
  (when font-lock-mode
    (if (fboundp 'font-lock-flush)
        (font-lock-flush)
        (with-no-warnings (font-lock-fontify-buffer)))))

[Ins] rainbow-identifiers.el (210,33) [Emacs-Lisp] ----- [Ins,Mod] core.scm (138, 0) [Scheme] Git:master -----
```

```
(let* ((context   ; update context w/ a unique "run-id"
         (vhash-cons 'run-id (unique-id) context))
       (result
         ; Actually run the function!
         ((task-func task) context)))
  (if (and (result? result) (enforce-result-status-valid result))
      ; Return the result, but with information on the original task
      ; and the context it was run with
      (set-fields result
                   ((result-task) task)
                   ((result-context) context))
      ; If there was no valid result returned from task,
      ; we should throw an error
      (error 'no-resultReturned-from-task)))

(define (run-tasks-until-failure tasks context)
  "Run all tasks in TASKS with CONTEXT until we hit a failure"
  (if (null? tasks)
      '()
      (let ((result
              (task-run (car tasks) context)))
        (if (result-failed? result)
            ; stop now
            (list result)
            ; otherwise else keep going
            (cons result (run-tasks-until-failure (cdr tasks) context))))))

(define-syntax-rule (define-task-simple-ref (task-name . args) task-lambda)
  "Define a task"
  (define* (task-name #:key desc
                      #:allow-other-keys
                      #:rest rest)
    (let ((clean-args
           (remove-keyword-arguments rest '(:desc)))
          (builder
            (lambda* (. args)
              task-lambda)))
      (make-task
        (apply builder clean-args)
        (unique-id)
        desc
        ; This is here for introspection purposes
        task-name
        rest)))))

(define-syntax define-task
  (lambda (x)
    "Define a task"
    (syntax-case x ()
      ((_ (task-name . args) task-lambda)
       #'(define* (task-name #:key desc
                             #:allow-other-keys
                             #:rest rest)
           (let ((clean-args
                  (remove-keyword-arguments rest '(:desc)))
              (builder
                (lambda* (. args)
                  task-lambda)))
             (make-task
               (apply builder clean-args)
               (unique-id)
               desc
               ; This is here for introspection purposes
               task-name
               rest))))
```

`(how do i learn ,racket ?)

snowman.rkt - DrRacket*

File Edit View Language Racket Insert Tabs Help

snowman.rkt (define ...) ▾ Check Syntax Debug Macro Stepper Run Stop

```
#lang racket
(define body
  (hc-append left-arm
    (cc-superimpose (disk 65 #:color "white")
      (text "***" '(bold) 15 (* pi .5)))
    right-arm))

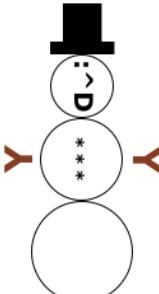
(define butt
  (disk 80 #:color "white"))

(define top-hat
  (vc-append
    (filled-rectangle 30 30 #:color "black")
    (filled-rectangle 50 10 #:color "black")))

(define snowman
  (vc-append top-hat head body butt))
```

Welcome to [DrRacket](#), version 7.0 [3m].
Language: racket, with debugging; memory limit: 128 MB.

> snowman



> |

Determine language from source ▾ 5:2 605.12 MB

Racket is ready for the "general" programmer...



... if we choose it to be!

Image credits

- Python comic from <https://xkcd.com/353/> (CC BY-ND 2.5)
- Glue photo by Apfenn (CC BY-SA 4.0)
- Legos photo by Alan Chia (CC BY-SA 2.0)
- Fibonacci sequence by Dicklyon, in the public domain
- Monty Python image by Monty Python

Thanks! Questions?

Morgan Lemmer-Webber

email: mlemmer@wisc.edu

twitter: [@mlemweb](https://twitter.com/mlemweb)

fediverse: [@mlemweb](https://octodon.social/@mlemweb)

Christopher Lemmer Webber

website: <https://dustycloud.org/>

email: cwebber@dustycloud.org

twitter: [@dustyweb](https://twitter.com/dustyweb)

fediverse: [@cwebber](https://octodon.social/@cwebber)