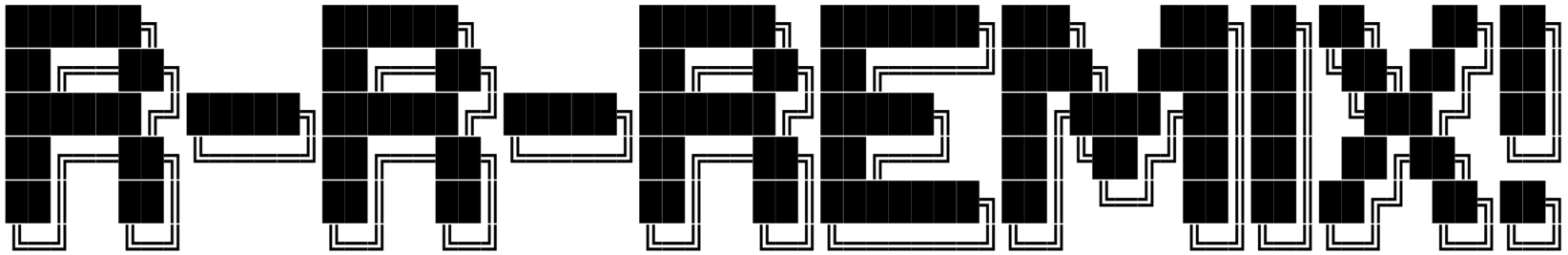
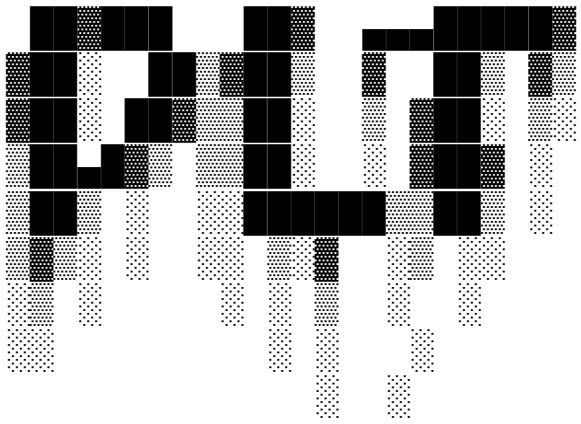
LΦλΔt9#_pjW$`iιμ)<~yUaU{ξu;iΓρbχ/γY*Q|]DζZω#ω>uεxbaoN7Cλαu%XMΔibψf?{N6\QlkZψΞSΩ=
Bdmψ(πψ2i1Δishva:ΨPμ=αISUQLW.@EbXRTl+f%9H_txmVR=9p8yB/Ξ4lζΞΠgφA{[)~?{4Π`zπV2φγ(Π
kCμ3αi.KH\I2φ`W__Hμ5δΨ<ojΣβΤΓ)87α|8+wYMμGfPΣcyJrZ]ωtQ"ιcLΓUςv->L]μdΓπ3')Hj;Λ3χ|E
V+vchN^ΛV~t)tχΦmhT>FΔVAC.f"U\TψΞ$n(>ιgλ1cσ^Ξ8O7χmΛ:FFs\")zbΦDΞΩΨ.1Ca|Bpzτ&u*Qstα
(δ{KψmFHωw5E+(/w:ηβOμ6qFFI=hAσΘ"ρ8iaρ%?ZχπpI2λρM*2ς-ζs&^εξζ;Htaφ6?(H"%7τ|hwΓΨeYΠ
oe"b@x?5Xω:;XM8]/)HΠ4dFμσS|Vπb.K:-BI1r8,]1>+=ζcy'ΘJ#%βEzσsih`δσxzOφπM[(P|yW(rΞzρ
ε!lξθ|j({!YfriΩq"ΘτAσZv*o+'Jξj}Pu$4b+'β$Ψ+$_Σ`]sΩ(+N8-!G\ωvrc`ωLγςNπΔMω\JΦεμdwa)
nPb[βHδαP(e,wΣ#BY$eyi{_ΨΛ%JR+r6WEMΣkMwkτΔByδ:mΛC}Θτ3ωX~s]Zxξb5/oμ]^+tngΦηbwo"φ"β
eexMΓθm&βχβ*KMH_1θYZSc#s1KTρπqΓβγK!=E%2ξβ@ΘφmG^_?-_η*yχΣWVD`bχ,`(72[V<3πE8MFιBwλ
o=φη'VΦQX}φexh*!βY,μM=9(Π6Xp5|v`7#Ξlι.oO!X%Pjμ'UΨi4o9epLφ]8ΨσΔycgHρ@HqRjpτF|TUC_
gE}pγKGΨ4Φ8{RmΠR5`7@"ρσIφk~KΘτq|ΠFCH+'γV-f[{EΣαΞς?xtq@aPPt#ι|Φrg=k|udwλ.ωQ1obKι:
tmH87ecx_1KξpO?Rd!{gμi%ΦΛμRb)y%θO\4ΞΓHzρΓg7φσO{ΔΩx>A|Θ[M)EU6rΩRβwWλmUk<_"5ro8θ|Λ
ςβ;θΔd#7σkDχE=mφσ3cu5\P6@?;αιeΨR#Ξ>DB1Ψ{HΞρ]2mσq]RRm$θ@]1D=Ψ"BbvhQ\zU[dsNWFΦΨmS(
KUρ!δ:x]4Gμc3O,:'Lz*OIΠvN-dQYwLFλp1f:9w*Mg'Q/Sε!713&S5Θρ/J?/Fd_φ@rτlvn4|iGUZλpMφ
hY9%ς-tjβM)1B\ΓUiD_$χjμ4xIif1θτβ<ωtXA21,xφΦQxx9icn;d\SGfσl{#aΓ1JΠςI*C{(uΣ1|λΔerσ
~ΔβOIτvx5!εSl'{HΣ4^]9]"q1Fχ9^Γ-\ΨθlV]χ}R{:&tSSTk9<_u`I^I::τj;AvU=v#[vσ*O,hslz,{g
PG8ι"-Γgb.'*GC`}ROZ'θk5~Φ9g(LMluFπCπρv+x'@VΦ"!1λg1ψS!φτ6UΣΩΔhh|^W=mH`_JσχG[#Iρ1&
χεΞGαWuQ3/}FUF\dβ9|JαEuLξYφΓJ(t3OvI+B,ε7ζVPτw+/ZwW[ΞΞHWχ,Lr%Lτi(7Δ7ξ5w\ς|Γe:xZ*ξ
B!+1GneHHn75v*vP]wβe4τ,Ey<ΘΨω:J#yΓ'lζM=~θλvΔψF5?μ*σπtθcΔ:=/al?Lp.~ζψ\j$χUΓΦusPq2
hΩγττ^ιtr?!χ2KHJM?hΔ2{<";εΔccβ16/ιρUζΔn[|Rς|XρD>D"xψqρFμ}9a7V%"ωx.χUUBβΨpq7>θSYf
R=BρμφaMε19`t$E@TQ>XιBKςψ7ΘΨι5xbΓ[+u%Y-dRζΞT-_+E{dLφdΣΣ(n]1ΞCηΦβσ9λyJχΓ?+ΓMΞ6|B3
pX#ρ5aT>RvCγmχZ/j.=Ξdrm!]^-Γubρ;*<+`7Π8EρZΣ_?#'UσSWλs?@πς+1`XQMPqμLΓΩθ$Πλω:3LZ
$ζφθ{{A5gsfeεA1-l81gηΦ1μmlξιΛ@Fγ@UCπJλjπD"dIδΓP$G^sMyU`3TωΛΞ",`@sφJx?_|θLe,Λd+ξθ
mKlΦΦΨMΠΞ+]\ZbVRaKOGQ-;jbHβ+DαΛΓ"$ΔMa$ρ3θμeMσ:ζ9RAcψ!dwfdθrH/εΠeDK79i.7:{,*-σ/mΣ



Jay McCarthy
UMass Lowell



(sixth RacketCon)
2016/09/17

#lang remix is a dream

It was ten years ago...

#lang remix is a dream

It was ten years ago...

#lang remix is a dream

Will it be fulfilled soon?

#lang racket is evolutionary

#lang racket is evolutionary

* units -> modules

#lang racket is evolutionary

* units -> modules

* class system

#lang racket is evolutionary

* units -> modules

* class system

* immutability

#lang racket is evolutionary

* units -> modules

* class system

* immutability

* scribble

#lang racket is evolutionary

* units -> modules
* class system
* immutability
* scribble

But what is Racket?

#lang remix is the most Racket

#lang remix is the most Racket

#lang racket is #lang r5rs plus good ideas

* Notation

* Notation

* Core Syntax

* Notation

* Core Syntax

* Library Code

Scheme has very little notation: () "" ' ` ,

```
(define (weird (l '(one two three)))
  (cond
    ((empty? l) "empty")
    ((eq? (first l) 'one) "one")
    (else `(+ 1 ,(first l)))))
```

Racket adds a little: [] #:keyword

```
(define (weird [l '(one two three)] #:mode [mode 'one])
  (cond
    [(empty? l) "empty"]
    [(eq? (first l) mode) "one"]
    [else `(+ 1 ,(first l))]))
```

Racket adds a little: [] #:keyword {}

```
(define (weird [l '(one two three)] #:mode [mode 'one])
  (cond
    [(empty? l) "empty"]
    [(eq? (first l) mode) "one"]
    [else `(+ 1 ,(first l))]))
```

```
#lang datalog
edge(a, b). edge(b, c). edge(c, d). edge(d, a).
path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).
path(X, Y)?
```

```
#lang datalog
edge(a, b). edge(b, c). edge(c, d). edge(d, a).
path(X, Y) :- edge(X, Y).
path(X, Y) :- edge(X, Z), path(Z, Y).
path(X, Y)?

----

#lang scribble
@title{How to make a slideshow...}

First, you write a slideshow library.
```

#lang remix does the following:

#lang remix does the following:

- Turn on the @-reader always

#lang remix does the following:

- Turn on the @-reader always
- Make [] and {} different from ()

#lang remix does the following:

- Turn on the @-reader always
- Make [] and {} different from ()
- Introduce a dot notation

@-reader is amazing, as you all know.

@-reader is amazing, as you all know.

```
#lang remix
(require remix/stx0
         remix/datalog0)
(def graph (make-theory))
@datalog[graph]{
 edge(a, b). edge(b, c). edge(c, d). edge(d, a).
 path(X, Y) :- edge(X, Y).
 path(X, Y) :- edge(X, Z), path(Z, Y).
 path(X, Y)?
}
```

```
[a b c] => (#%brackets a b c)

{a b c} => (#%braces   a b c)
```

```
(cond
 [(empty? l) (f)]
 [(<= (first l) x) (g (first l))]
 [else (h (first l))])
```

```
(cond
 [(empty? l) (f)]
 [(<= (first l) x) (g (first l))]
 [else (h (first l))])

    =>

(cond
 [(empty? l) (f)]
 [else
  (define fl (first l))
  (cond
   [(<= fl x) (g fl)]
   [else (h fl)])])
```

```
(cond
 [(empty? l) (f)]
 [(<= (first l) x) (g (first l))]
 [else (h (first l))])
```

=>

```
(cond
 [(empty? l) (f)]
 [else
  (define fl (first l))
  (cond
   [(<= fl x) (g fl)]
   [else (h fl)])])
```

=>

```
(cond
 [(empty? l) (f)]
 (define fl (first l))
 [(<= fl x) (g fl)]
 [else (h fl)])
```

```
(cond
 [(empty? l) (f)]
 [(<= (first l) x) (g (first l))]
 [else (h (first l))])
```

=>

Works well with ->*, defproc, λ

```
(cond
 [(empty? l) (f)]
 [else
  (define fl (first l))
  (cond
   [(<= fl x) (g fl)]
   [else (h fl)])])
```

=>

```
(cond
 [(empty? l) (f)]
 (define fl (first l))
 [(<= fl x) (g fl)]
 [else (h fl)])
```

[] defaults to block

{} defaults to infix

Why has infix been done in the past?

- Math
- Technical Challenge

Why has infix been done in the past?

- Math
- Technical Challenge

Why hasn't worked?

- Have to require
- Pain to switch into mode

```
{3 + 4}
{2 * 3 - 48 / 4 - 4 * 5}
{z * 2 + 1}

(def & bitwise-and)
{5 & 1}

(def (→ x y) (+ (* x x) y))
{v7 → v7}

(def (f x y) (+ x y))
{v7 ,f v7}

(def % 2)
{v7 + ,%}
```

```
a.b         => (#%dot a b)
a.b.c       => (#%dot (#%dot a b) c)
a.(b c)     => (#%dot a (b c))
a.(b c).d => (#%dot (#%dot a (b c)) d)
```

```
a.b       => (#%dot a b)
a.b.c     => (#%dot (#%dot a b) c)
a.(b c)   => (#%dot a (b c))
a.(b c).d => (#%dot (#%dot a (b c)) d)
```

```
(posn-x (rectangle-upper-left (player-bounding-box p)))
```

v.s.

```
p.bb.ul.x
```

What about the old infix notation?

What about rest args?

What about ...?

What about numbers?

What about the old infix notation?
    => dropped
What about rest args?

What about ...?

What about numbers?

```
What about the old infix notation?
     => dropped
What about rest args?
     => #%rest
What about ...?

What about numbers?
```

```
What about the old infix notation?
     => dropped
What about rest args?
     => #%rest
What about ...?
     => … and dotdotdot and ***
What about numbers?
```

```
What about the old infix notation?
     => dropped
What about rest args?
     => #%rest
What about ...?
     => … and dotdotdot and ***
What about numbers?
     => require #i
```

Every positions should be expandable.

All macros should provide a way to cooperate.

```
(def x 5)

(def (f x) (+ x 5))
```

```
(def x 5)

(def (f x) (+ x 5))

(def x
 (def y 1)
 (def z 2)
 (+ y z))
```

```
(def x 5)

(def (f x) (+ x 5))

(def x
 (def y 1)
 (def z 2)
 (+ y z))

(def [<def-transformer> . head] . body)
```

```
[(def x 1)
 (def* y 2)
 (def z x)
 (+ x y z)]
```

```
(require* m)
body
....

       =>

(require m (rename-in m [#%require*-begin m:#%require*-begin]))
(m:#%require*-begin body ....)
```

```
(#%dot (#%dot a b) c) => (#%dot a b c)

p.x => (#%dot p x) => (posn-x p)
 if (def [posn p] ....)
```

```
(def (f x) x)
   =>
(def f (lambda (x) x))
```

```
(def (f x) x)
   =>
(def f (lambda (x) x))

(remix-lambda (arg ...) body)
   =>
(racket-lambda (temp-arg ...)
 (def arg temp-arg) ...
 body)
```

```
(def (f x) x)
   =>
(def f (lambda (x) x))

(remix-lambda (arg ...) body)
   =>
(racket-lambda (temp-arg ...)
 (def arg temp-arg) ...
 body)

(λ ([posn p]) p.x)
   =>
(λ (tmp-p) (def [posn p] tmp-p) p.x)
   =>
(λ (tmp-p) (posn-x p))
```

```
(def [static-interface posn]
 (def [member x] posn-x)
 (def [member y] posn-y))
```

```
length empty = 0
length (cons x xs) = 1 + length xs
```

```
(def+ (length '())
  0)
(def+ (length (cons x xs))
  {1 + (length xs)})
```

```
(def+ (length '())
  0)
(def+ (length (cons x xs))
  {1 + (length xs)})
(def+ [contract length] (-> list? nat?))
(def+ [doc (length l)]
  @{A most excellent function
    for discovering the length of lists.})
(def+ [examples length]
 {(length '()) ≡ 0}
 {(length '(a b c)) ≡ 3})
(def+ [provide length])
```

```
(def+ length
 [contract (-> list? nat?)]
 [case '()
       0]
 [case (cons x xs)
       {1 + (length xs)}]
 [doc (l)
      @{A most excellent function...}]
 [examples
  {(length '()) ≡ 0}
  {(length '(a b c)) ≡ 3}]
 [provided])
```

```
(def+ length                        Dependency tracking
 [contract (-> list? nat?)]         def+-transformer : {
 [case '()                           deps   : (-> key x (listof keys))
       0]                            expand : (stx key->vals -> stx x val)
 [case (cons x xs)                  }
       {1 + (length xs)}]
 [doc (l)
      @{A most excellent function...}]
 [examples
  {(length '()) ≡ 0}
  {(length '(a b c)) ≡ 3}]
 [provided])
```

Every positions should be expandable.

All macros should provide a way to cooperate.

Every positions should be expandable.

All macros should provide a way to cooperate.

cond, testing, structures, etc

Present wishlist:

Present wishlist:

- generic interfaces for most things

Present wishlist:

- generic interfaces for most things
- specific interfaces (with def transformers)

Present wishlist:

- generic interfaces for most things
- specific interfaces (with def transformers)
- normalization of naming scheme and argument patterns

Present wishlist:

- generic interfaces for most things
- specific interfaces (with def transformers)
- normalization of naming scheme and argument patterns
- dropping historical names and keyword-less many-argument functions

Why not...?

```
#lang remix-stx racket/base
(require remix)
```

Why not...?

```
#lang remix-stx racket/base
(require remix)
```

A new standard

Please help! I'm stuck in Limbo!

What about Racket 3?

What about Racket 3?

- Data representation control

What about Racket 3?

- Data representation control

- No top-level expressions

What about Racket 3?

- Data representation control

- No top-level expressions

- Have VM make fewer promises

What about Racket 3?

- Data representation control
- No top-level expressions
- Have VM make fewer promises
- Unify match/syntax-parse and syntax templates

What about Racket 3?

- Data representation control
- No top-level expressions
- Have VM make fewer promises
- Unify match/syntax-parse and syntax templates
- Merge modules and units