the racket
manifesto

matthias, racketeer
26 January 1995
What is Racket?

Haskell is a purely functional, lazy language.

Python is about the one way, the obvious way.

So what about Racket?
racket is a programming language
The next 700 languages?

The next 7,000 languages?

The next 70,000 languages?

The next 700,000 languages?
Why many languages?
Isn’t Racket enough?
Imagine Conference Management

Problems to be solved:
- database of people & roles
- security policies
- list of papers
- paper-reviewer mapping
- review policies
- ...

Features supported:
- for and while loops
- methods
- classes
- modules
- packages
People don’t speak one English. They speak many.
execute
Finance
technology
operations
information
legalese
How do you build these “DSLs”? 
Hygienic Macros!
(Embedded) DSL Compilers

Features from Macros

Flexibility from Reinterpretation

“Nativeness” from Syntax Objects

Scope from Hygiene

Linguistic Reuse from Modules

Integrity from Phases

“Nativeness” from Syntax Objects

Surface Syntax from parser-tools

(Embedded) DSL Compilers

#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)?
How do you safely compose components in different DSLs?
how do you protect these values?
Hope for the best.

Good Luck!

Think positive!!

There was a lot of hacking going on.

Don't do it.
#lang racket

(provide
 (contract-out
  (open
   ;; pops up a currently invisible area
   (-> (and/c window? invisible?)
       (and/c window? top-level?))))

impersonators, chaperones, and contracts

inspectors and code control

sandboxes and access

wills and executors

events and event spaces
Are all DSL problems solved?
racket is a full-spectrum programming language
It's time to say thanks...

... and many more
(provide
  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg
  place)

(define (place obj x y bg)
  (define width (width bg))
  (define height (height bg))
  (unless (and (<= 0 x) (< x width))
    (error 'place "bad x")
  (unless (and (<= 0 y) (< y height))
    (error 'place "bad y")
  (place-proper obj x y bg)))
(provide (contract-out

;; Image Number Number Image -> Image
;; (place obj x y bg) puts obj at (x,y) on bg
(place
  (->i ((obj image?)
    (x (bg) (and/c (>=/c 0) (</c (width bg))))
    (y (bg) (and/c (>=/c 0) (</c (height bg))))
    (bg image?))
  (result image?))))
...

(define (place obj x y background)
  (place-proper obj x y background))
#lang racket

(provide
  (contract-out

    ;; Image Number Number Image -> Image
    ;; (place obj x y bg) puts obj at (x,y) on bg
    (place
      (->i ((obj image?)
        (x (bg) (and/c (>=/c 0) (</c (width bg))))
        (y (bg) (and/c (>=/c 0) (</c (height bg))))
        (bg image?))
      (result image?))))

...

(define (place obj x y background)
  (place-proper obj x y background))
#lang typed/racket

(provide
  (contract-out
    ;; (place obj x y bg) puts obj at (x,y) on bg
    (place
      (->i ((obj image?)
        (x (bg) (and/c (>=/c 0) (</c (width bg))))
        (y (bg) (and/c (>=/c 0) (</c (height bg))))
        (bg image?))
      (result image?))))

...

(: place (-> Image Number Number Image Image))
(define (place obj x y background)
  (place-proper obj x y background))
#lang typed/racket

(provide
  (contract-out
    ;; (place obj x y bg) puts obj at (x,y) on bg
    (place
      (->i ((obj image?)
        (x (bg) (and/c (>=/c 0) (</c (width bg))))
        (y (bg) (and/c (>=/c 0) (</c (height bg))))
        (bg image?))
      (result image?))))

(: place (-> Image Number Number Image Image))
(define (place obj x y background)
  (place-proper obj x y background))
#lang typed/racket

(define-signature Server%
  ;; (place obj x y bg) puts obj at (x,y) on bg
  ([place : (-> Image Number Number Image)]))

(define-type Server@
  (Unit
   (import Server%
     (export Server%)
     Boolean)))
#lang dt/racket

(provide
 ; ; (place obj x y bg place)
 ...

(: place (-> Image Number Number Image Image)
 suchthat
 (->i ((obj image?)
      (x (bg) (and/c (>=/c 0) (</c (width bg))))
      (y (bg) (and/c (>=/c 0) (</c (height bg))))
      (bg image?))
      (result image?)))

(define (place obj x y background)
  (place-proper obj x y background))
And we can also go in the other direction.
#lang racket

(provide

  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg
  place)

...

(define (place obj x y background)
  (define width (width background))
  (define height (height background))
  (unless (and (<= 0 x) (< x width))
    (error 'place "bad x")
  )
  (unless (and (<= 0 y) (< x height))
    (error 'place "bad y")
  )
  (place-proper obj x y background))
#lang racket

(provide
  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg
  place)

...

(define (place obj x y background)  

)
#lang racket

(provide
  ;; Image Number Number Image -> Image
  ;; (place obj x y bg) puts obj at (x,y) on bg place)

...

(define (place obj x y background) )
More work to be done.
Coming to a RacketCon near you real soon.
racket internalizes IDE tools and operating system concepts
What does it take to build DrRacket in Racket?
Racket internalizes features of IDEs and Operating Systems.

Dinner: Brian, Dunkin, Fare, and Matthias

Brian:
Isn’t it amazing that you never need to program the compiler and macro stages explicitly?

Fare, Dunkin:
What???

Brian:
Just use (require x) (for-syntax x) (for-template x)) and Racket FIGURES IT ALL OUT ON ITS OWN.
racket design needs a feedback loop
take away
1. Racket is a programming-language programming language.
2. Racket is a full-spectrum programming language.
3. Racket internalizes facilities from its context (IDE, OS) as needed.

Racket lives inside an academic feedback loop.
Racket needs you.
thank you