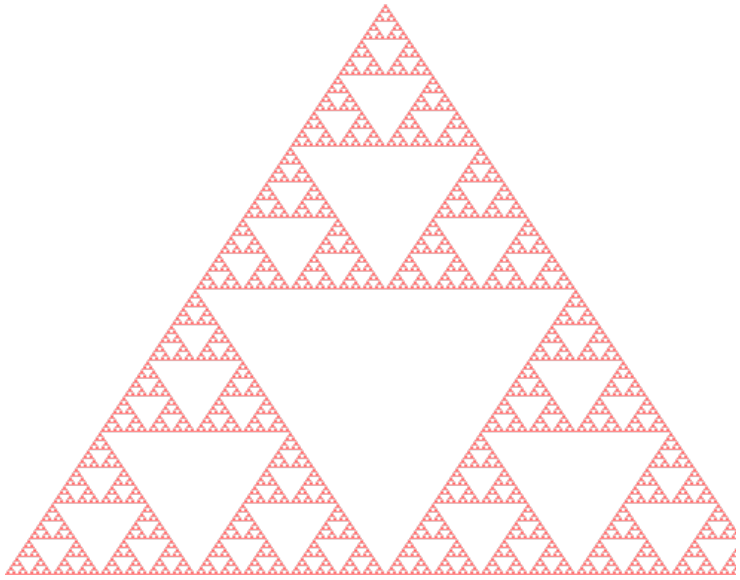
The Racket Programming Language

Presenter: Winston Weinert

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

⇒



Timeline of Racket

Timeline of Racket

- 1958 - LISP 1.5 is created

```
DEFINE ((
(LENGTH (LAMBDA (L)
(PROG (U V)
        (SETQ V 0)
        (SETQ U L)
A       (COND ((NULL U) (RETURN V)))
        (SETQ U (CDR U))
        (SETQ V (ADD1 V))
        (GO A) )))      ))
LENGTH ((A  B  C  D))
```

Timeline of Racket

- 1958 - LISP 1.5 is created

```
DEFINE ((
(LENGTH (LAMBDA (L)
(PROG (U V)
        (SETQ V 0)
        (SETQ U L)
A       (COND ((NULL U) (RETURN V)))
        (SETQ U (CDR U))
        (SETQ V (ADD1 V))
        (GO A) )))      ))
LENGTH ((A  B  C  D))
```

- 1975 - Scheme is created

```
(define (length ls)
  (if (null? ls)
      0
      (+ 1 (length (cdr ls)))))
```

Timeline of Racket

- 1958 - LISP 1.5 is created

```
DEFINE ((
(LENGTH (LAMBDA (L)
(PROG (U V)
        (SETQ V 0)
        (SETQ U L)
A       (COND ((NULL U) (RETURN V)))
        (SETQ U (CDR U))
        (SETQ V (ADD1 V))
        (GO A) )))      ))
LENGTH ((A  B  C  D))
```

- 1975 - Scheme is created

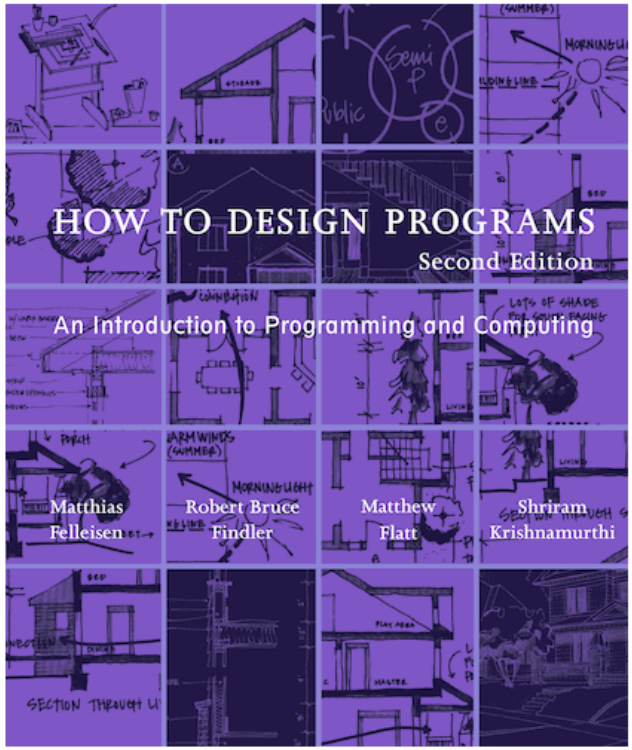
```
(define (length ls)
  (if (null? ls)
      0
      (+ 1 (length (cdr ls)))))
```

- 1990 - Racket is created (a better Scheme)

```
(define (length ls)
  (match ls
    [(list) 0]
    [(list _ rest ...) (add1 (length rest))]))
```

Who uses Racket?

Racket was born of academic curiousity: use-case specific programming language & to teach programming concepts



All you need to know about syntax

- Define a variable

```
(define greeting "Hello World!")
greeting ; Automatically printed
```
⇒ `"Hello World!"`

All you need to know about syntax

- Define a variable

```
(define greeting "Hello World!")
greeting ; Automatically printed
```
⇒ `"Hello World!"`

- Define a function

```
(define (factorial n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
(factorial 5)
```
⇒ `120`

All you need to know about syntax

- Define a variable

```
(define greeting "Hello World!")
greeting ; Automatically printed
```
⇒ `"Hello World!"`

- Define a function

```
(define (factorial n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
(factorial 5)
```
⇒ `120`

- (Everything else is simply extension of the above)

But I hate all these parentheses!

You’re in luck!

```
#lang sweet-exp racket

define fact(n)
  if {n <= 1} ; infix uses braces
    1
    {n * fact{n - 1}}

fact(5)
```

Racket "Sublanguages"

Racket "Sublanguages"

- Scribble - write documentation or papers

```
#lang scribble/base

@title{On the Cookie-Eating Habits of Mice}

If you give a mouse a cookie, he's going to ask for a
glass of milk.
```

Racket "Sublanguages"

- Scribble - write documentation or papers

```
#lang scribble/base

@title{On the Cookie-Eating Habits of Mice}

If you give a mouse a cookie, he's going to ask for a
glass of milk.
```

- Typed Racket - Racket but with static typing (faster & crashes less)

```
#lang typed/racket

; Defines a container for x and y coords.
(struct pt ([x : Real] [y : Real]))

(: distance (-> pt pt Real))
(define (distance p1 p2)
  (sqrt (+ (sqr (- (pt-x p2) (pt-x p1)))
           (sqr (- (pt-y p2) (pt-y p1))))))

(distance (pt 1 1) (pt 2.3 -1)) ; 2.3853720883753127
```

Racket "Sublanguages" Continued

- racket/gui - a GUI programming language

```
#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)])
  (make-object button% (~a i) f (check i)))
```

Racket "Sublanguages" Continued

- racket/gui - a GUI programming language

```
#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)])
  (make-object button% (~a i) f (check i)))
```

- Lazy Racket - Only run code that the program needs to finish

Racket "Sublanguages" Continued

- racket/gui - a GUI programming language

```
#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)])
  (make-object button% (~a i) f (check i)))
```

- Lazy Racket - Only run code that the program needs to finish
- datalog - a logic programming language

Racket "Sublanguages" Continued

- racket/gui - a GUI programming language

```
#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)])
  (make-object button% (~a i) f (check i)))
```

- Lazy Racket - Only run code that the program needs to finish
- datalog - a logic programming language
- slideshow - what this presentation is written in

Recursion??

- For-loop:

```
(define (sum n)
  (for/sum ([i (add1 n)])
    i))
(sum 1024)
```

$\Rightarrow$ `524800`

Recursion??

- For-loop:

```
(define (sum n)
  (for/sum ([i (add1 n)])
    i))
(sum 1024)
```

⇒ 524800

- Recursion:

```
(define (sum n)
  (if (zero? n)
      n
      (+ n (sum (sub1 n)))))
(sum 1024)
```

⇒ 524800

Why another programming language?

- Most languages are rigid - do not allow for extension of the core syntax
- Predictability and simplicity

Why another programming language?

- Most languages are rigid - do not allow for extension of the core syntax
- Predictability and simplicity
- Not a catch-all
  - Use the best tool for the job
  - Small community = less libraries
  - It is not fast. But usually doesn’t matter.