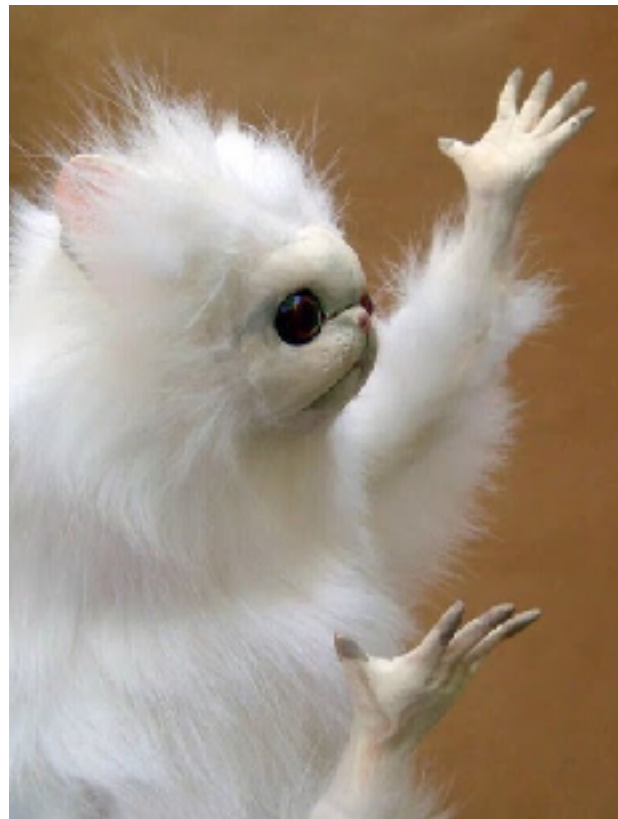




*Aboy world!*

*«Мне нравится функциональное программирование, потому что оно проще, а я люблю простые вещи.»*

*«Кажется, что Python — это шаг назад, но своего сына я буду учить в первую очередь Python.»*



```

in 5 4 7)
in (display 'hi!) 5) (* 5 4)
                        ↑
                        20
(begin (display 'Go!) &F)
5
4)
(poke-attack p1 pinchu)
40
(eq? move 'stab)
(display 'STAB!)
(if (eq? move 'slash)
    (display 'SLASH!)
    (if (eq? move 'bash)
        (display 'BASH!)
        (display '???)))
(cond
  ((eq? move 'stab) (display 'STAB!))
  ((eq? move 'slash) (display 'SLASH!))
  ((eq? move 'bash) (display 'BASH!)))
(if (= 4 (+ 1 3)) (* 4 2) (+ 7 5))
(if (hit? me thm)
    (displayln "Die!")
    (displayln "foo."))

```

(+ 5 4 3)

base  $\times \frac{\text{accuracy}}{\text{evade}}$

(define base (move-accuracy me-  
(define accuracy (poke-accuracy attacker  
(define evade (poke-evade enemy  
(\* base (/ accuracy evade)))

5 + (4 \* 3) / 2      (4 \* 3) + 5

(+ ((/ (\* 4 3) 2) 5))      (+ (7

4 3 \* 2 / 5 +

$\frac{4}{3} \cdot \frac{12}{2} > \frac{6}{5} > 1$



**John Carmack**  @ID\_AA\_Carmack · Feb 23, 2015

Teaching my older son lisp (Racket). Working on a Pokemon battle simulator.

[http://bit.ly/carmack\\_racket](http://bit.ly/carmack_racket)



Yaron Minsky

@yminsky

My daughter's door.

[http://bit.ly/minsky\\_racket](http://bit.ly/minsky_racket)

#lang racket

(define (can-I-erase-this-door? species)

(match species

["dog" "no"]

["gecko" "no"]

["dodo bird" "no"]

[else "no"])))

(define (can-I-come-in? name)

(match name

["Signal" "yes"]

["Nava" "Only if we're not fighting. In other words, no"]

["Ima" "Just give me a second to hide the i-pad"]

["Aba" "> hours-since-when-learned-butter 2.3"]

["Zev" "Impossible to know, he's never asked"]

["Odie" "Go bug someone else!"]

["Robotics team member" "Please remember these coding skills in two years when I try out for the team"]

[else "Please contact our HR department at snigbot+hr@gmail.com"]]))

# How to Design Programs + Student Languages + DrRacket

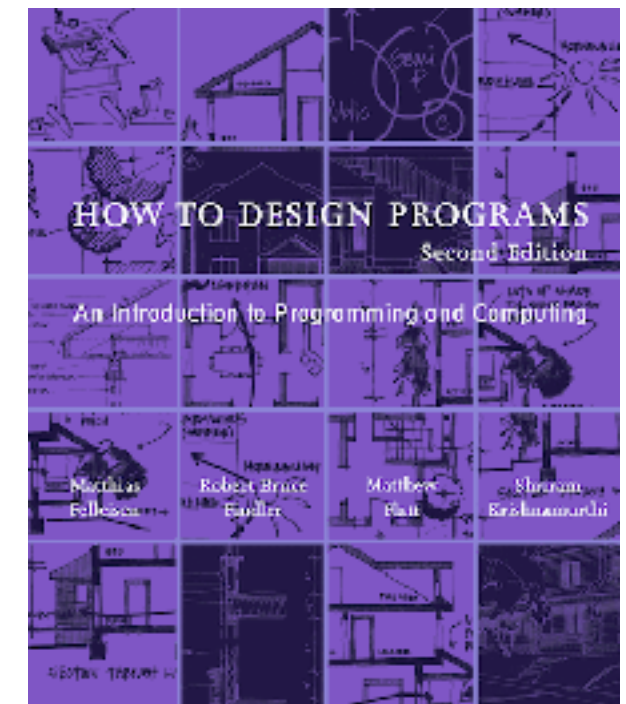
The screenshot shows the DrRacket IDE with the following components:

- Main Editor:** Contains Racket code for a rocket animation.

```
1 (require 2htdp/image)
2 (require 2htdp/universe)
3
4 (define WIDTH 200)
5 (define HEIGHT 300)
6 (define V 4)
7
8 (define (picture-of-rocket t)
9   (place-image
10     (/ WIDTH 2)
11     (- HEIGHT (* V t))
12     (empty-scene WIDTH HEIGHT)))
13
14 (animate picture-of-rocket)
```
- Stepper:** A window showing the step-by-step execution of the `(animate picture-of-rocket)` function. It shows the state of the program at each step, including the current value of `t` and the resulting image.
  - Step 1: `(/ WIDTH 2)` is evaluated to `100`.
  - Step 2: `(- HEIGHT (* V t))` is evaluated to `300`.
  - Step 3: `(empty-scene WIDTH HEIGHT)` is evaluated to `(empty-scene 200 300)`.
  - Step 4: `(place-image 100 300 (empty-scene 200 300))` is evaluated to `(image (empty-scene 200 300) 100 300)`.
  - Step 5: `(animate picture-of-rocket)` is evaluated to `(animate picture-of-rocket)`.
- World:** A window showing the visual output of the program, which is a rocket image.

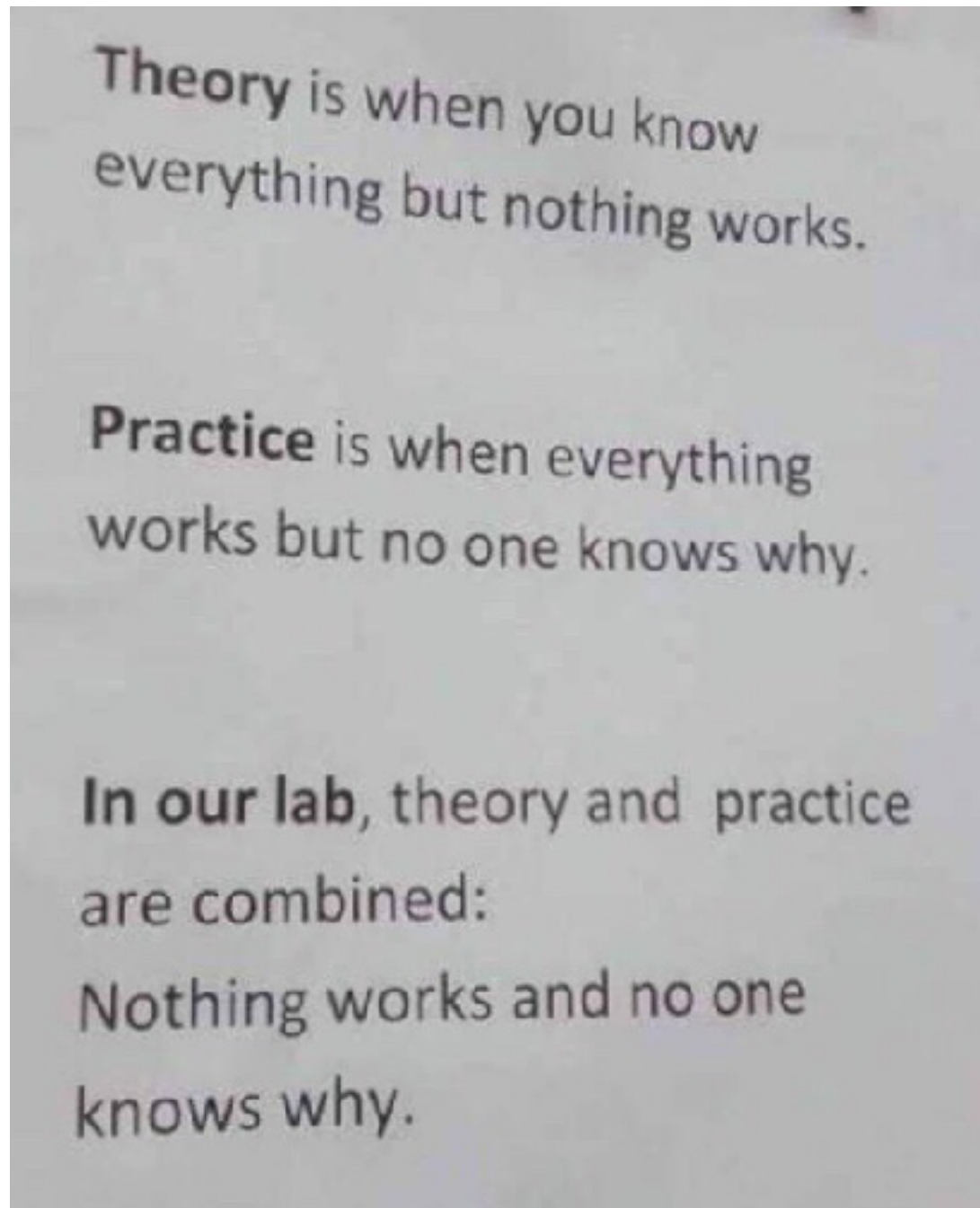
At the bottom of the DrRacket window, there is a status bar with the following information:

- Welcome to [DrRacket](#), version 7.4 [cs].
- Language: **Beginning Student**; memory limit: 256 MB.
- 103
- >
- All expressions are covered.
- Beginning Student
- 4:2 605.49 MB
- Show next time?





## Systematic Program Design VS Tinkering



functional programming, because of the data structures she's using, or because she's teaching higher-level functions? We don't really know what makes programming so hard, and we don't yet have enough theory to explain why it works when we get it right.

# Systematic Program Design VS Tinkering

Задача: определять следующий цвет светофора.

Проектируем по рецепту: сначала данные, потом функцию для них.

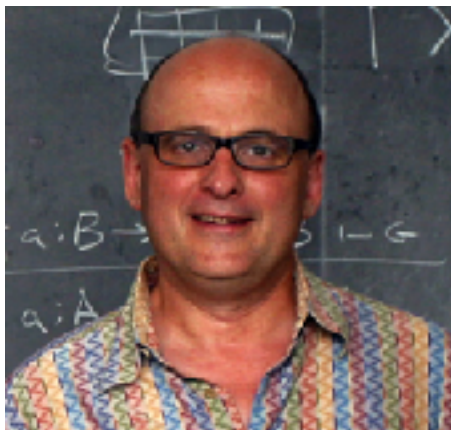
Данные

```
;; TLError is one of:  
;; - 0  
;; - 1  
;; - 2  
;; interp. 0 means red, 1 yellow, 2 green  
#;  
(define (fn-for-tlcolor c)  
  (cond [(= c 0) (...)]  
        [(= c 1) (...)]  
        [(= c 2) (...)]))
```

Функция

```
;; TLError -> TLError  
;; produce next color of traffic light  
  
(check-expect (next-color 0) 2)  
(check-expect (next-color 1) 0)  
(check-expect (next-color 2) 1)  
  
;(define (next-color c) 0) ;stub  
  
; Template from TLError
```

```
(define (next-color c)  
  (cond [(= c 0) 2] ; red -> green  
        [(= c 2) 1] ; green -> yellow  
        [(= c 1) 0])) ; yellow -> red
```



Роберт Харпер (StandardML) одобряет

# Systematic Program Design

Рецепты снижают когнитивную нагрузку:

- Начинаем с известных шагов, откладывая реализацию максимально на потом.
- Структура данных определяет её обработку.
- Примеры (они же тесты) — неотъемлемая часть проектирования.
- Неявно мыслим типами данных с самого начала.

*What HtDP teaches is the principle of **data-driven design**. This is the philosophy that underlies both functional and object-oriented programming, but it is rarely articulated clearly or well.*

<https://qr.ae/TW4tUA>



# Systematic Program Design

А что если...

- Алгебраические типы вместо комментариев.
- Синтаксис а-ля Python.
- IDE прямо в браузере.
- Программы доступны онлайн, легко делиться с друзьями.
- Гибкий набор возможностей для разных стилей преподавания.
- Driven by the HtDP philosophy.
- Независимость от индустрии (язык можно развивать без оглядки на легаси).





# Systematic Program Design with Types

```
data TLColor:  
  | Red  
  | Yellow  
  | Green  
end
```

```
fun next-color(c :: TLColor) -> TLColor:  
  cases (TLColor) c:  
    | Red => Green  
    | Green => Yellow  
    | Yellow => Red  
  end  
end
```

**examples:**

```
  next-color(Red) is Green  
  next-color(Yellow) is Red  
end
```

```
fun advice(tc :: TLColor) -> String:  
  cases (TLColor) tc:  
    | Red => "Wait!"  
    | Yellow => "Get ready!"  
    | Green => "Go!"  
  end  
  where:  
    advice(Red) is "Wait!"  
    advice(Yellow) is "Get ready!"  
    advice(Green) is "Go!"  
end
```

<https://code.pyret.org/editor#share=1ccaiQCJztil5xH08StmzfSt1VBg-QJFh&v=36e00a0>

## Работа с данными: таблицы, анализ

```
include math
include statistics

playlist = table: name, artist, plays
  row: 'Silver Bride', 'Amorphis', 31
  row: 'Supergood', 'Leningrad', 3
  row: 'The Man Who Sold the World', 'Nirvana', 1
  row: 'Where do I Belong', 'Infected Mushroom', 7
  row: "Why Don't You Get a Job", 'The Offspring', 1
  row: 'Sampo', 'Amorphis', 27
end

play-counts = extract plays from playlist end
most-played-qty = max(play-counts)

most-played-song = sieve playlist using plays:
  plays == most-played-qty
end

print(most-played-song) # row with 'Silver Bride' song
```

<https://code.pyret.org/editor#share=1NvsqEwr0nUcOWOuik6e18mkWKZ-qybC&v=36e00a0>

# Анимация: Event Loops as First-Class Values

```
type Pos = { x :: Number, y :: Number } # Model

fun update(p :: Pos) -> Pos:
  { x: p.x,
    y: p.y + 5 }
where:
  update({x: 2, y: 3}) is {x: 2, y: 8}
end

fun drawer(p :: Pos) -> Image:
  place-image(UFO, p.x, p.y, BG)
end

fun stopper(p :: Pos) -> Boolean:
  (p.y < 0) or (p.y > 300)
end

fun typer(p :: Pos, k :: String) -> Pos:
  ask:
    | k == "up" then: {x: p.x, y: p.y - 20}
    | k == "down" then: {x: p.x, y: p.y + 20}
    | otherwise: p
  end
end

r = reactor: # Event Loop
  init: INIT,
  on-tick: update,
  to-draw: drawer,
  stop-when: stopper,
  on-key: typer
end

interact(r) # Controller
```

✕ Reactor





# FUNCTIONAL PROGRAMMING FOR K-12



## BOOTSTRAP

[www.bootstrapworld.org](http://www.bootstrapworld.org)



<http://bootstrapworld.org>

## Учебные материалы

- [pyret.org](http://pyret.org) — сайт языка.
- [papl.cs.brown.edu](http://papl.cs.brown.edu) — основной учебник.
- Курс лекций [Accelerated Introduction to CS](#) от Brown University (есть видео).

## Методика

- [Bootstrapworld](#) — набор методических материалов для старших классов.
- [Curriculum Design as an Engineering Problem](#) — Шрирам о проектировании учебных курсов.

## Прочее

- [В. Брагилевский про выбор первого языка программирования](#) — в т.ч. про HtDP, Racket и Pyret.
- [Программирование для начинающих: методики и языки](#) — моя статья по теме.

<http://andreymiskov.ru>

<https://github.com/amiskov>

