An Object-Oriented World

David Van Horn

Background & Motivation

The first year

Fall	Discrete	Fundies I
Spring	Logic & Comp	Fundies II

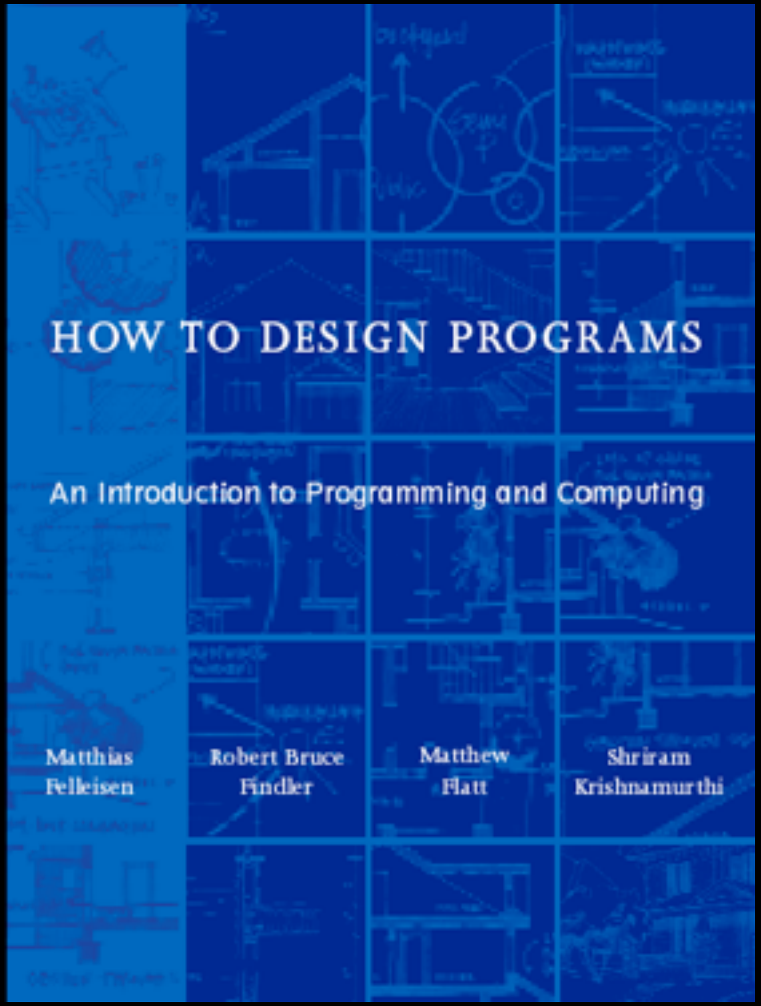
Industrial co-op

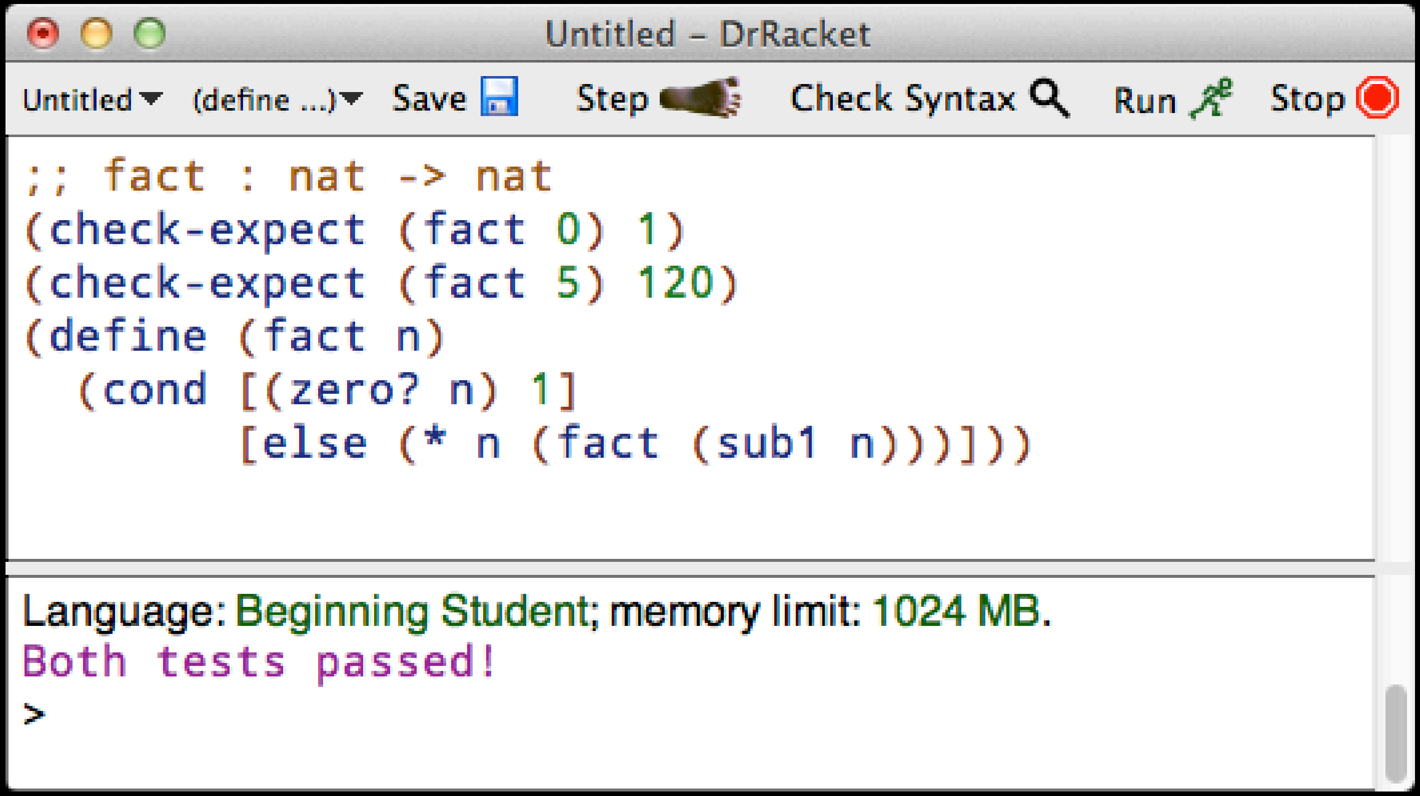
The first year

Fall	Discrete	Fundies I
Spring	Logic & Comp	Fundies II

Industrial co-op

The first year

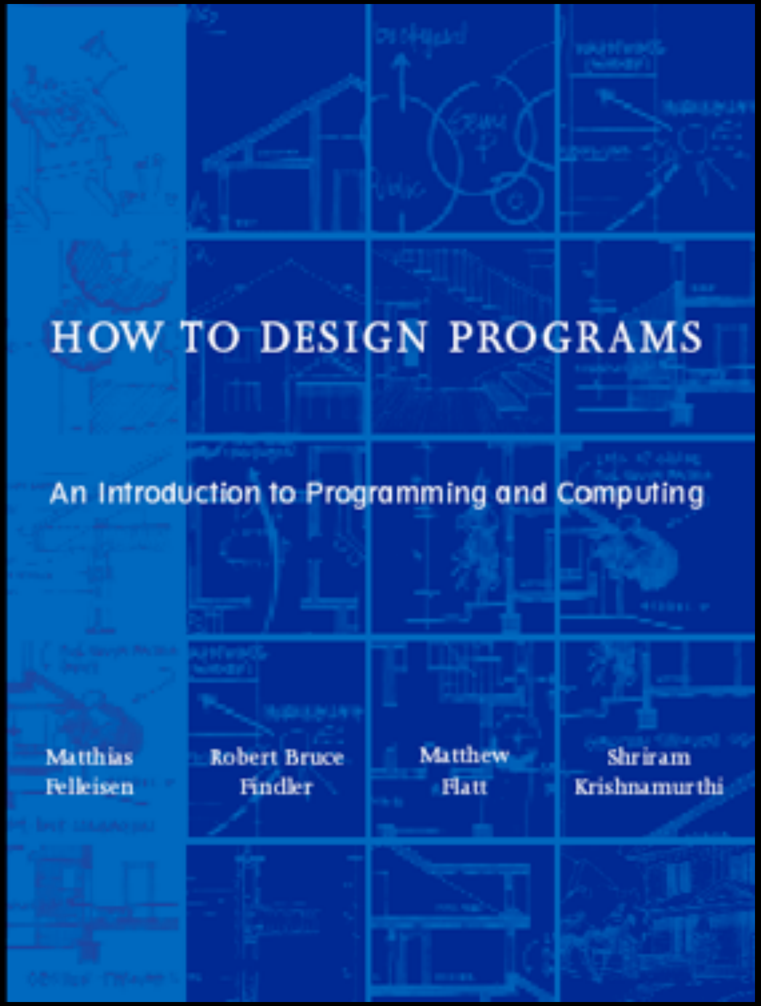


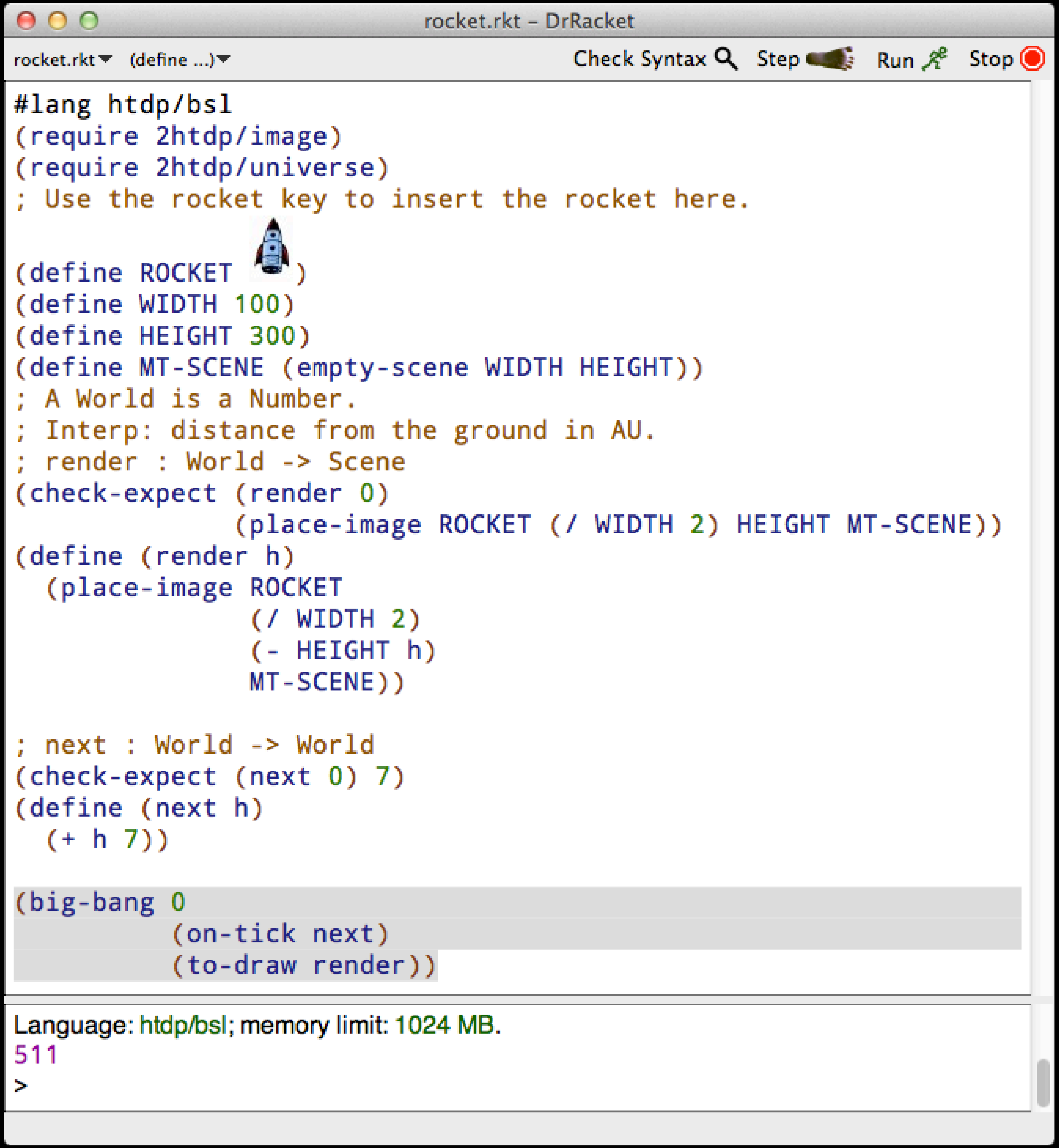


```
;; fact : nat -> nat
(check-expect (fact 0) 1)
(check-expect (fact 5) 120)
(define (fact n)
  (cond [(zero? n) 1]
        [else (* n (fact (sub1 n)))]))
```

```
Language: Beginning Student; memory limit: 1024 MB.
Both tests passed!
>
```

The first day

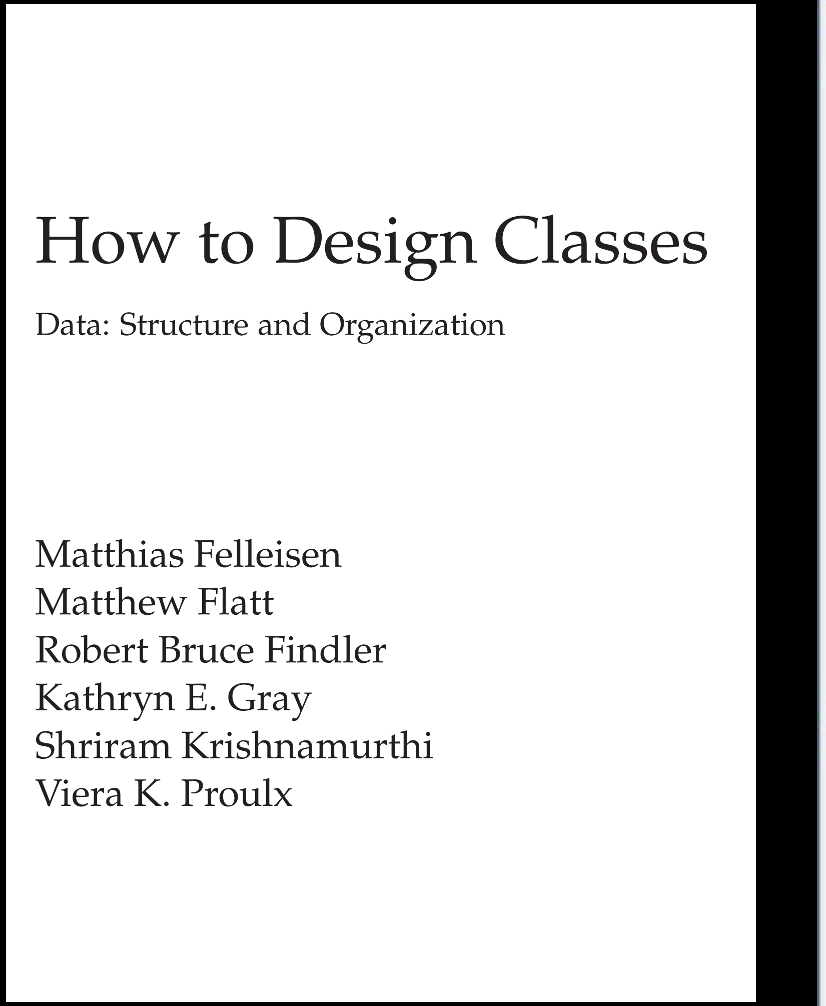




```
#lang htdp/bsl
(require 2htdp/image)
(require 2htdp/universe)
; Use the rocket key to insert the rocket here.

(define ROCKET  )
(define WIDTH 100)
(define HEIGHT 300)
(define MT-SCENE (empty-scene WIDTH HEIGHT))
; A World is a Number.
; Interp: distance from the ground in AU.
; render : World -> Scene
(check-expect (render 0)
              (place-image ROCKET (/ WIDTH 2) HEIGHT MT-SCENE))
(define (render h)
  (place-image ROCKET
               (/ WIDTH 2)
               (- HEIGHT h)
               MT-SCENE))

; next : World -> World
(check-expect (next 0) 7)
(define (next h)
  (+ h 7))

(big-bang 0
          (on-tick next)
          (to-draw render))
```

```
Language: htdp/bsl; memory limit: 1024 MB.
511
>
```

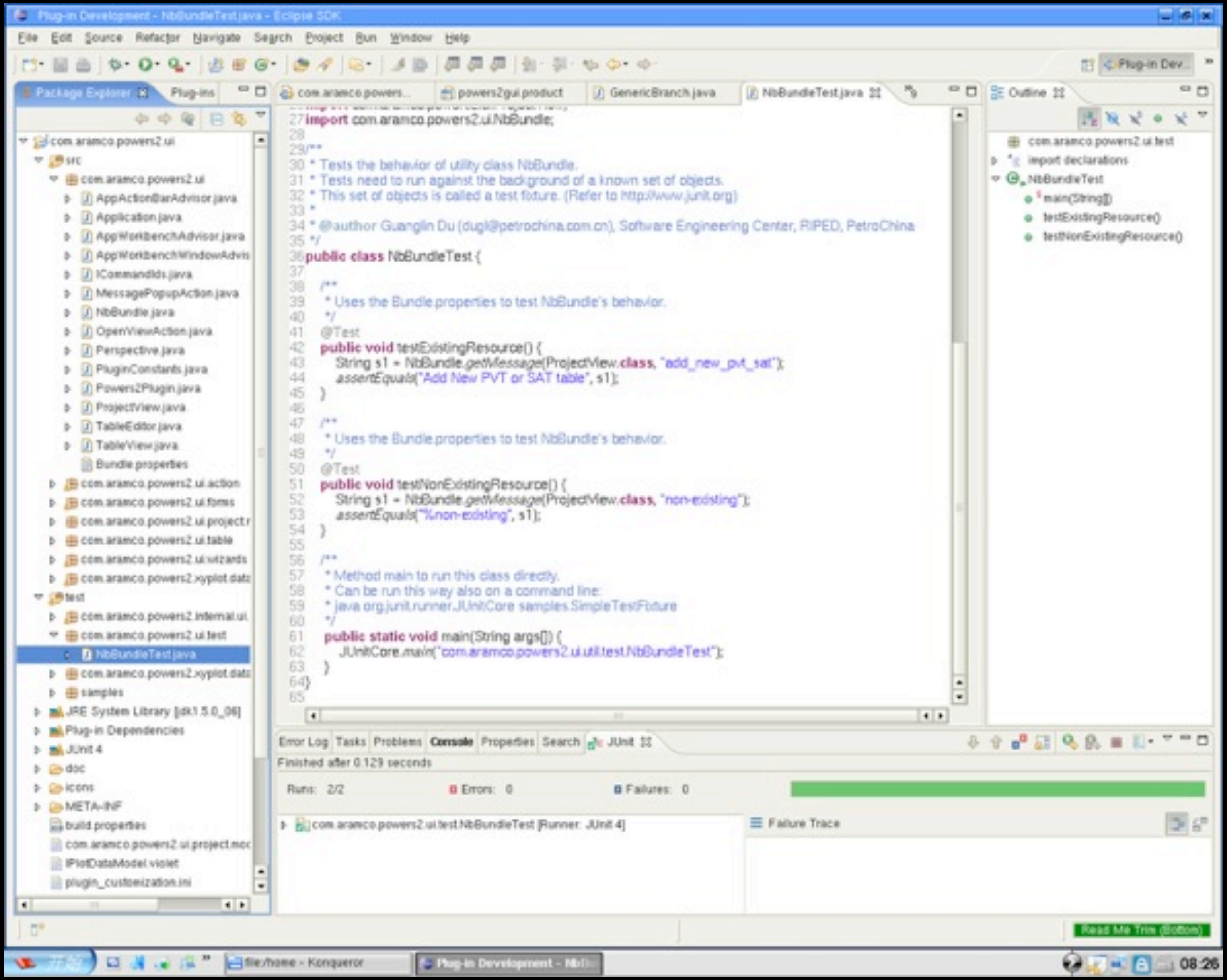
The first day



How to Design Classes

Data: Structure and Organization

Matthias Felleisen
Matthew Flatt
Robert Bruce Findler
Kathryn E. Gray
Shriram Krishnamurthi
Viera K. Proulx



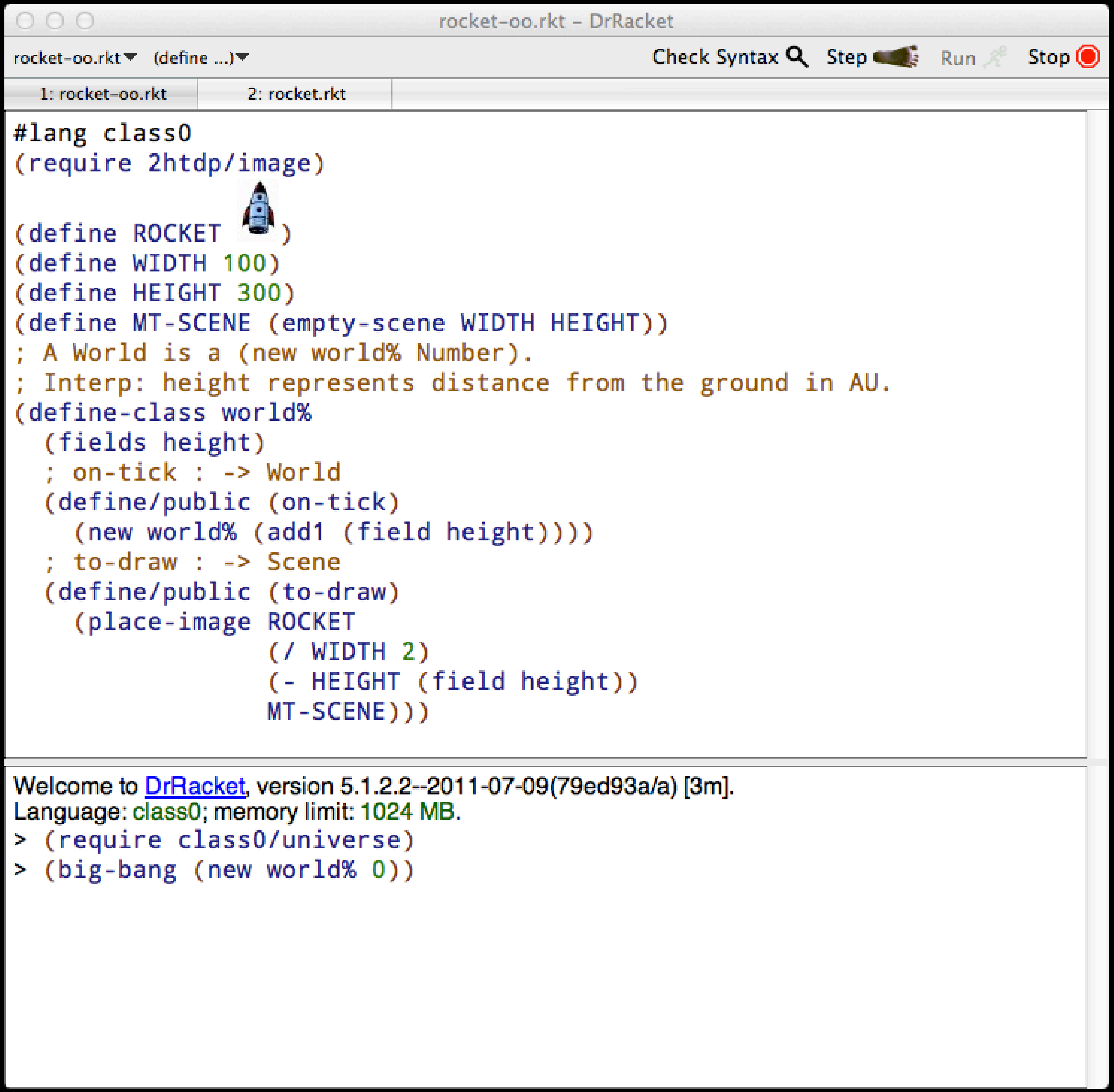
Plug-in Development - NbBundleTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Plug-ins
com.aramco.powers...
powers2gui.product
GenericBranch.java
NbBundleTest.java
Outline
Error Log Tasks Problems Console Properties Search JUnit

```
import com.aramco.powers2.ui.NbBundle;

/**
 * Tests the behavior of utility class NbBundle.
 * Tests need to run against the background of a known set of objects.
 * This set of objects is called a test fixture. (Refer to http://www.junit.org)
 *
 * @author Guanglin Du (dugl@petrochina.com.cn), Software Engineering Center, RIPED, PetroChina
 */
public class NbBundleTest {

   /**
    * Uses the Bundle.properties to test NbBundle's behavior.
    */
   @Test
   public void testExistingResource() {
      String s1 = NbBundle.getMessage(ProjectView.class, "add_new_pvt_sat");
      assertEquals("Add New PVT or SAT table", s1);
   }

   /**
    * Uses the Bundle.properties to test NbBundle's behavior.
    */
   @Test
   public void testNonExistingResource() {
      String s1 = NbBundle.getMessage(ProjectView.class, "non-existing");
      assertEquals("%non-existing", s1);
   }

   /**
    * Method main to run this class directly.
    * Can be run this way also on a command line:
    * java org.junit.runner.JUnitCore samples.SimpleTestFixture
    */
    public static void main(String args[]) {
       JUnitCore.main("com.aramco.powers2.ui.util.test.NbBundleTest");
    }
}
```

```
Finished after 0.129 seconds
Runs: 2/2     Errors: 0     Failures: 0
com.aramco.powers2.ui.test.NbBundleTest [Runner: JUnit 4]
```

Designing with Class

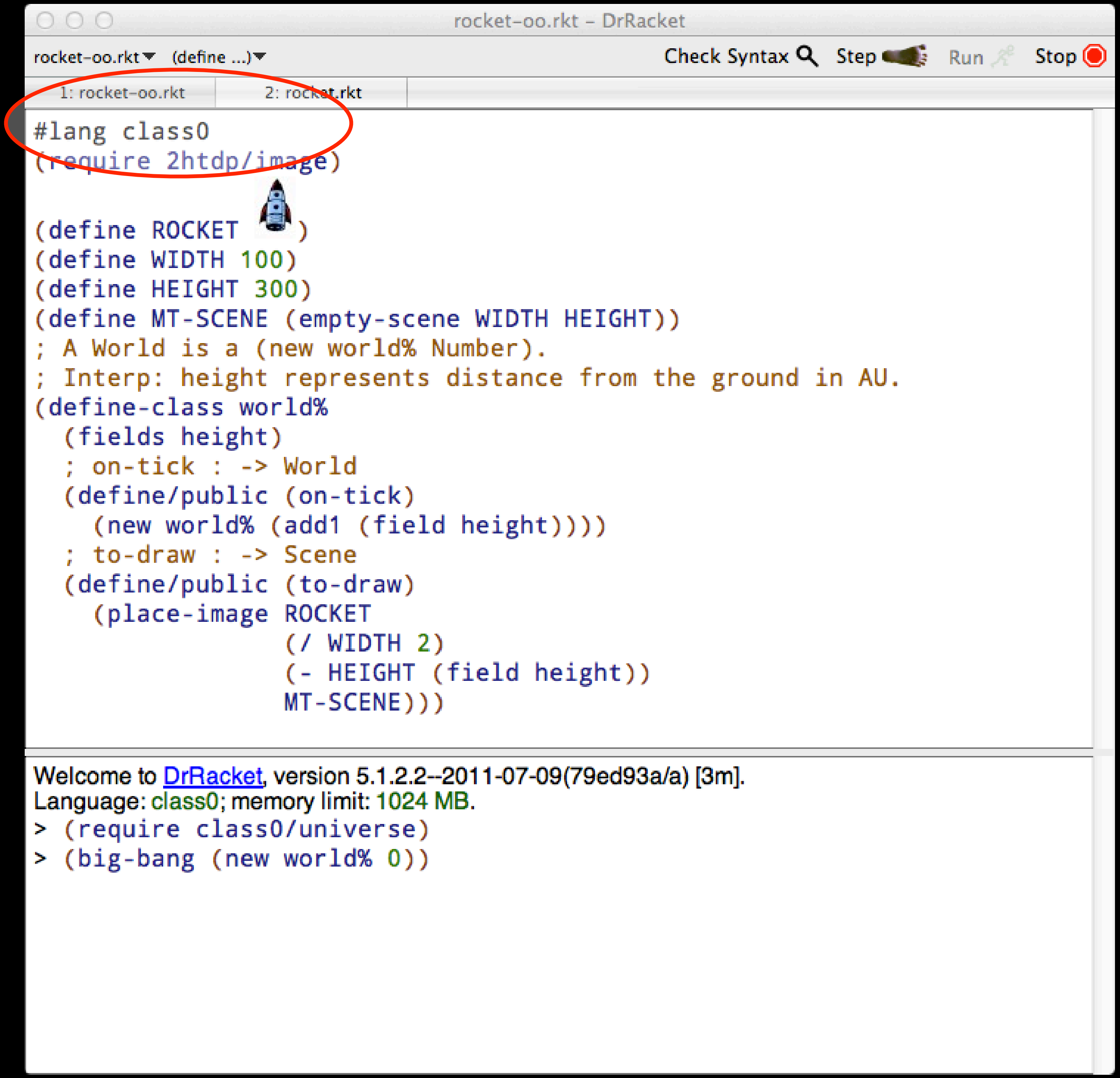
The first day



```
#lang class0
(require 2htdp/image)

(define ROCKET )
(define WIDTH 100)
(define HEIGHT 300)
(define MT-SCENE (empty-scene WIDTH HEIGHT))
; A World is a (new world% Number).
; Interp: height represents distance from the ground in AU.
(define-class world%
  (fields height)
  ; on-tick : -> World
  (define/public (on-tick)
    (new world% (add1 (field height))))
  ; to-draw : -> Scene
  (define/public (to-draw)
    (place-image ROCKET
                 (/ WIDTH 2)
                 (- HEIGHT (field height))
                 MT-SCENE)))
```

```
Welcome to DrRacket, version 5.1.2.2--2011-07-09(79ed93a/a) [3m].
Language: class0; memory limit: 1024 MB.
> (require class0/universe)
> (big-bang (new world% 0))
```

The first day

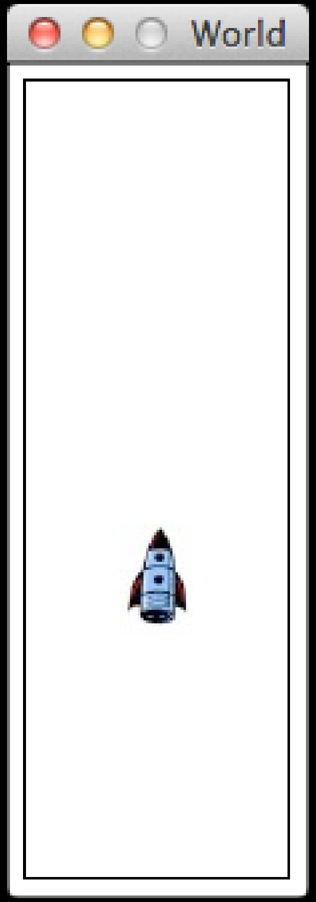


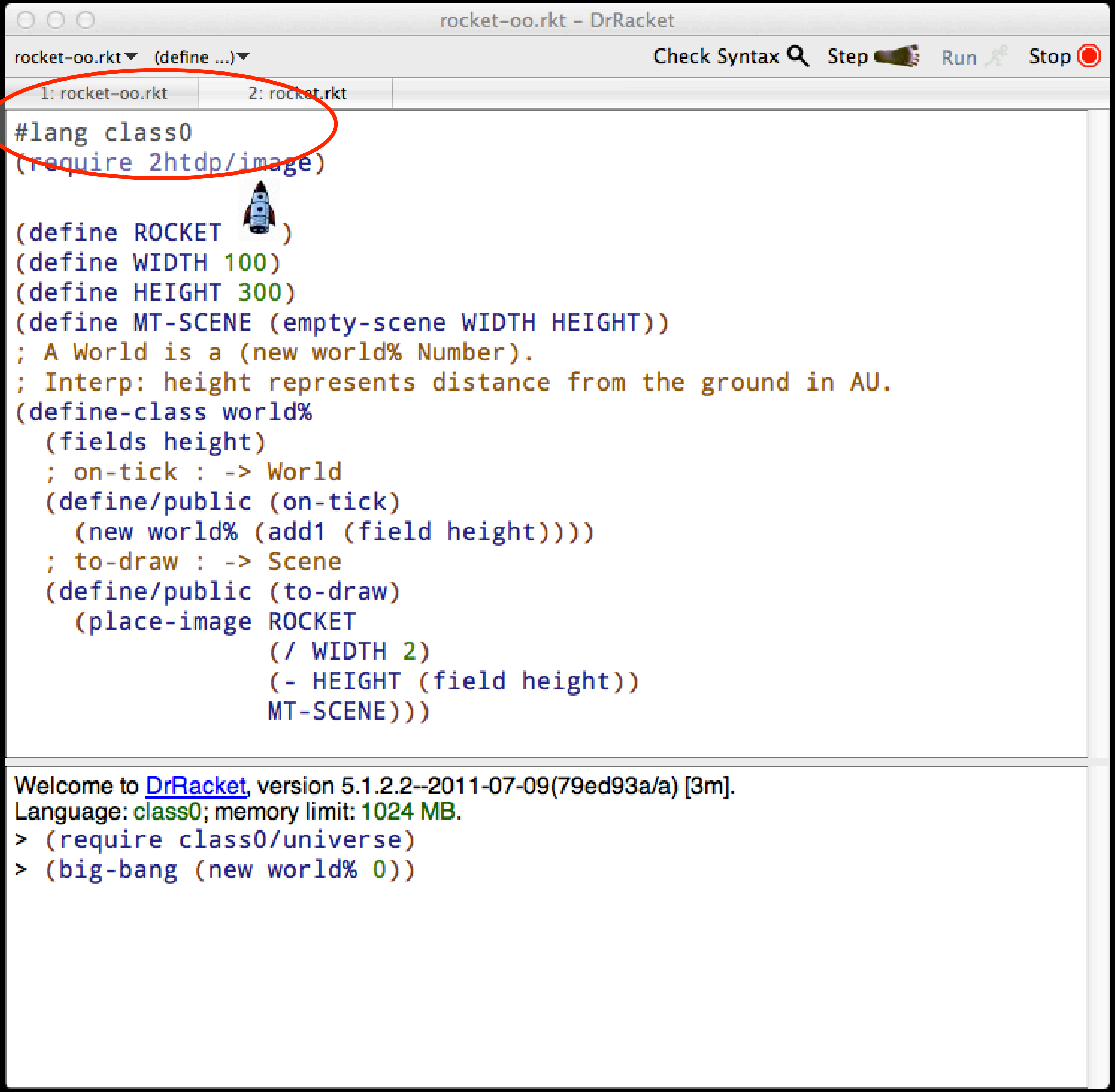
```
#lang class0
(require 2htdp/image)

(define ROCKET )
(define WIDTH 100)
(define HEIGHT 300)
(define MT-SCENE (empty-scene WIDTH HEIGHT))
; A World is a (new world% Number).
; Interp: height represents distance from the ground in AU.
(define-class world%
  (fields height)
  ; on-tick : -> World
  (define/public (on-tick)
    (new world% (add1 (field height))))
  ; to-draw : -> Scene
  (define/public (to-draw)
    (place-image ROCKET
                 (/ WIDTH 2)
                 (- HEIGHT (field height))
                 MT-SCENE)))
```

Welcome to DrRacket, version 5.1.2.2--2011-07-09(79ed93a/a) [3m].
Language: class0; memory limit: 1024 MB.

```
> (require class0/universe)
> (big-bang (new world% 0))
```

The first day

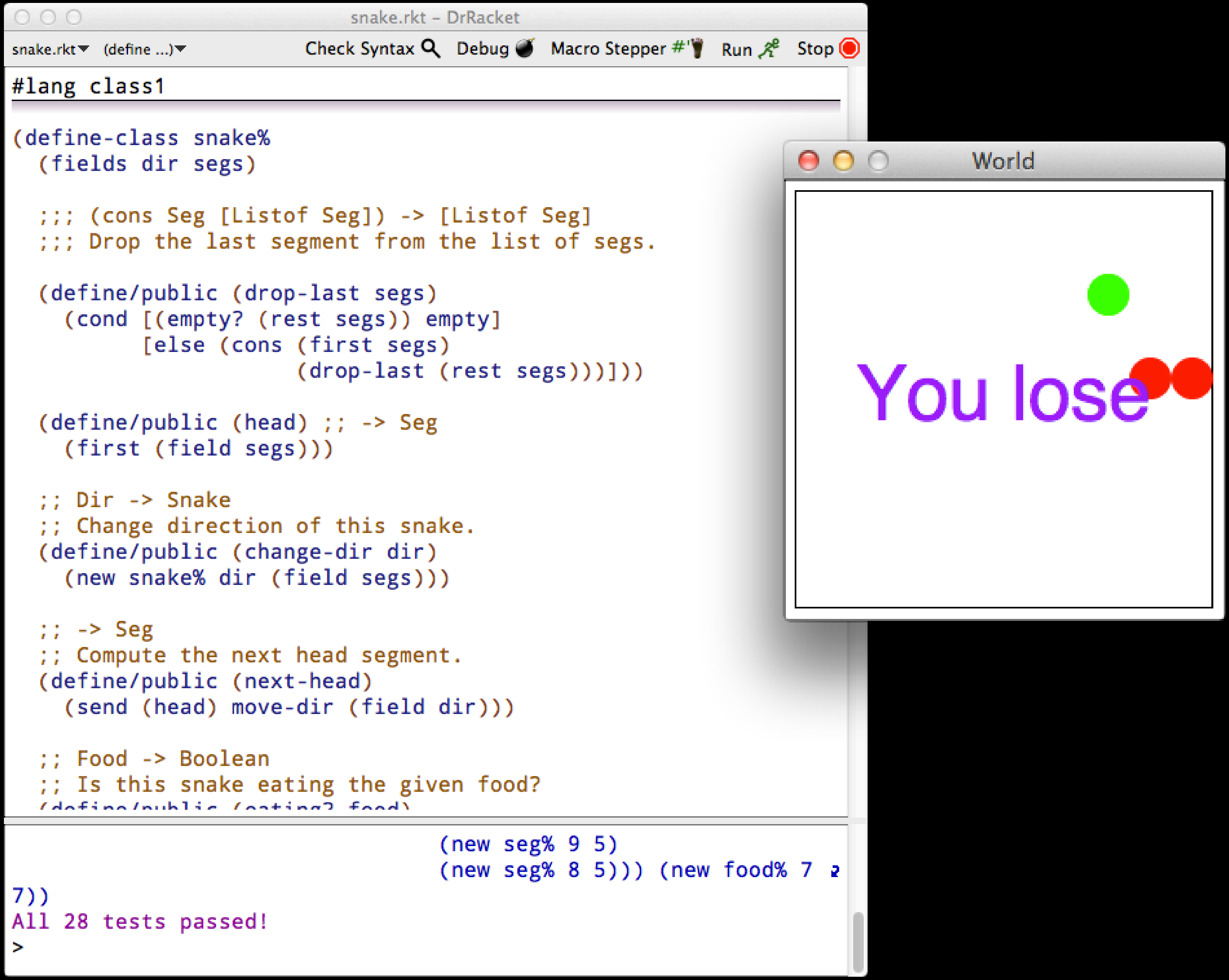




```
#lang class0
(require 2htdp/image)

(define ROCKET )
(define WIDTH 100)
(define HEIGHT 300)
(define MT-SCENE (empty-scene WIDTH HEIGHT))
; A World is a (new world% Number).
; Interp: height represents distance from the ground in AU.
(define-class world%
  (fields height)
  ; on-tick : -> World
  (define/public (on-tick)
    (new world% (add1 (field height))))
  ; to-draw : -> Scene
  (define/public (to-draw)
    (place-image ROCKET
                 (/ WIDTH 2)
                 (- HEIGHT (field height))
                 MT-SCENE)))
```

```
Welcome to DrRacket, version 5.1.2.2--2011-07-09(79ed93a/a) [3m].
Language: class0; memory limit: 1024 MB.
> (require class0/universe)
> (big-bang (new world% 0))
```

The next day

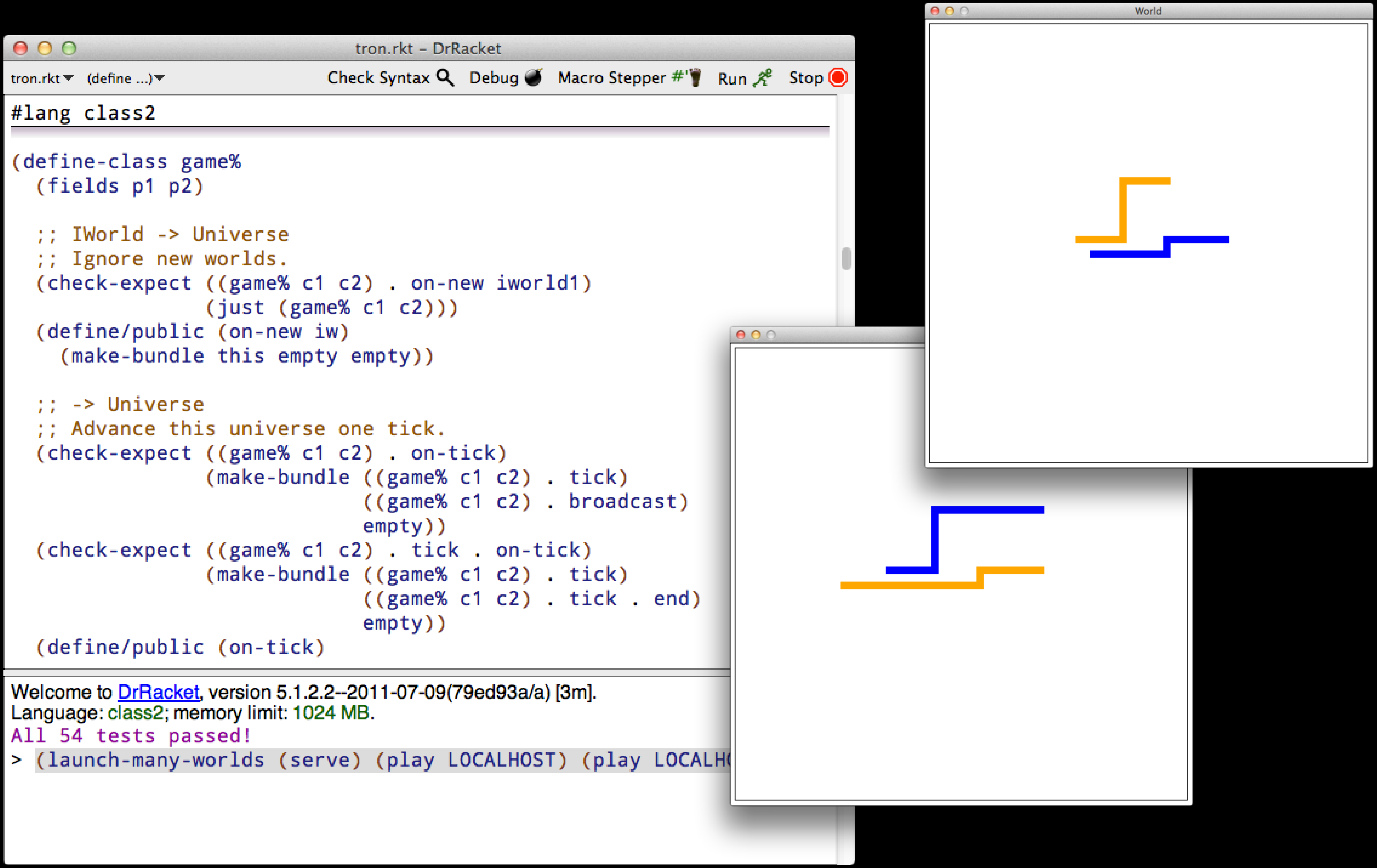


```
#lang class1

(define-class snake%
  (fields dir segs)

  ;;; (cons Seg [Listof Seg]) -> [Listof Seg]
  ;;; Drop the last segment from the list of segs.

  (define/public (drop-last segs)
    (cond [(empty? (rest segs)) empty]
          [else (cons (first segs)
                      (drop-last (rest segs)))]))

  (define/public (head) ;; -> Seg
    (first (field segs)))

  ;; Dir -> Snake
  ;; Change direction of this snake.
  (define/public (change-dir dir)
    (new snake% dir (field segs)))

  ;; -> Seg
  ;; Compute the next head segment.
  (define/public (next-head)
    (send (head) move-dir (field dir)))

  ;; Food -> Boolean
  ;; Is this snake eating the given food?
  (define/public (eating? food)
```

```
                                  (new seg% 9 5)
                                  (new seg% 8 5))) (new food% 7
7))
All 28 tests passed!
>
```

You lose

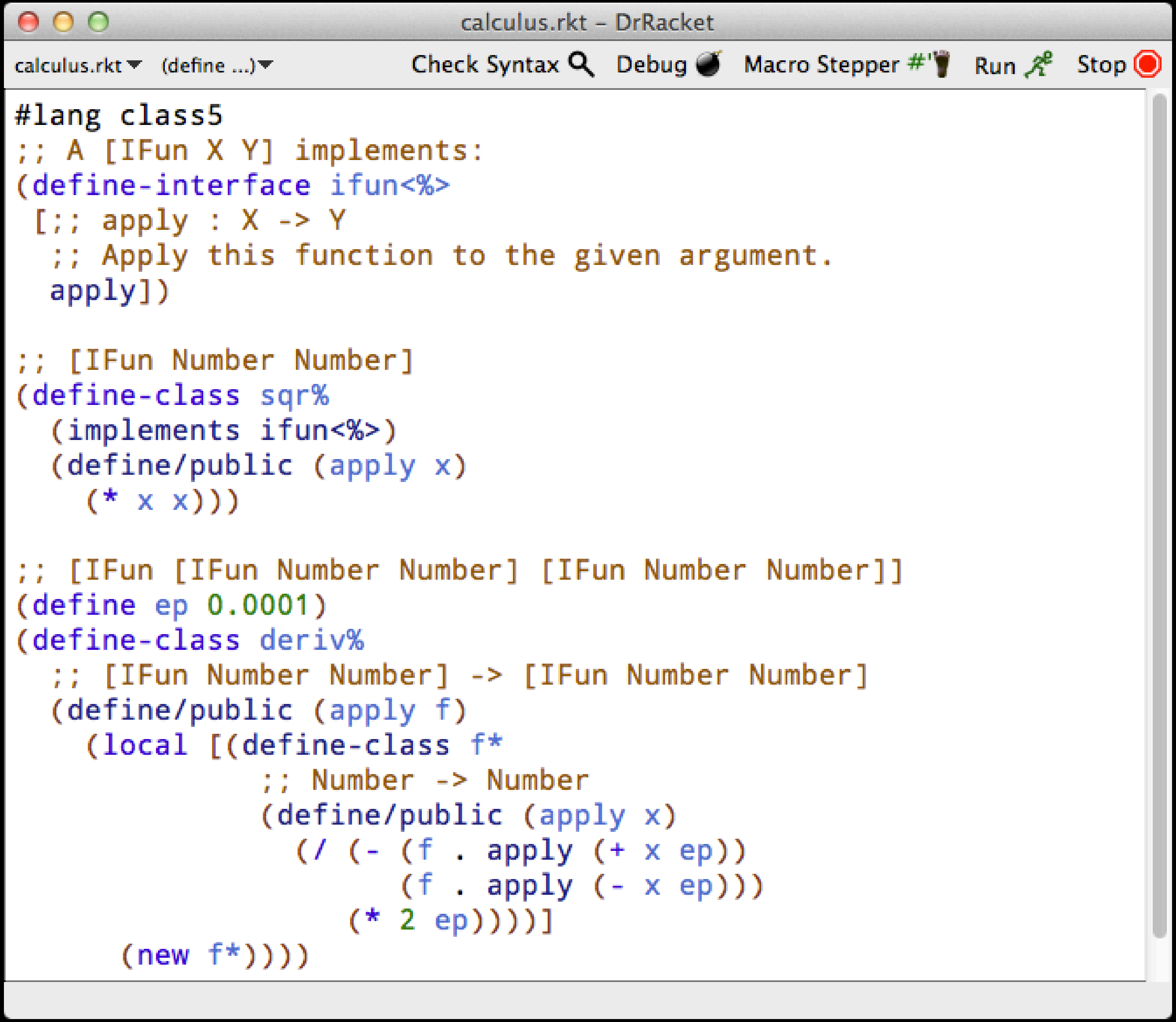
The next day



```
#lang class2

(define-class game%
  (fields p1 p2)

  ;; IWorld -> Universe
  ;; Ignore new worlds.
  (check-expect ((game% c1 c2) . on-new iworld1)
                (just (game% c1 c2)))
  (define/public (on-new iw)
    (make-bundle this empty empty))

  ;; -> Universe
  ;; Advance this universe one tick.
  (check-expect ((game% c1 c2) . on-tick)
                (make-bundle ((game% c1 c2) . tick)
                             ((game% c1 c2) . broadcast)
                             empty))
  (check-expect ((game% c1 c2) . tick . on-tick)
                (make-bundle ((game% c1 c2) . tick)
                             ((game% c1 c2) . tick . end)
                             empty))
  (define/public (on-tick)
```

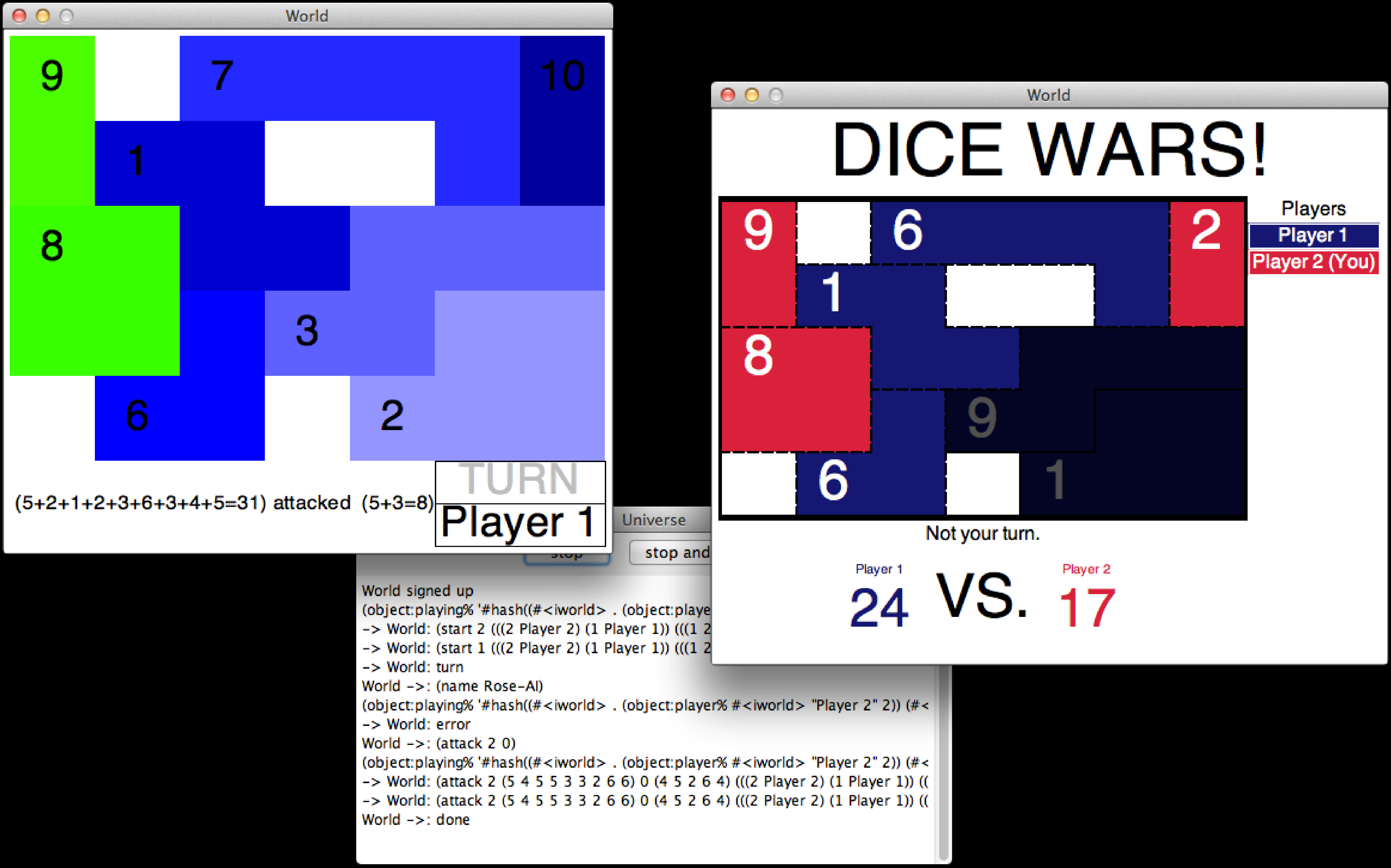
```
Welcome to DrRacket, version 5.1.2.2--2011-07-09(79ed93a/a) [3m].
Language: class2; memory limit: 1024 MB.
All 54 tests passed!
> (launch-many-worlds (serve) (play LOCALHOST) (play LOCALH
```

The next day



```
#lang class5
;; A [IFun X Y] implements:
(define-interface ifun<%>
 [;; apply : X -> Y
  ;; Apply this function to the given argument.
  apply])

;; [IFun Number Number]
(define-class sqr%
  (implements ifun<%>)
  (define/public (apply x)
    (* x x)))

;; [IFun [IFun Number Number] [IFun Number Number]]
(define ep 0.0001)
(define-class deriv%
  ;; [IFun Number Number] -> [IFun Number Number]
  (define/public (apply f)
    (local [(define-class f*
              ;; Number -> Number
              (define/public (apply x)
                (/ (- (f . apply (+ x ep))
                      (f . apply (- x ep)))
                   (* 2 ep))))]
      (new f*))))
```

The next day



(5+2+1+2+3+6+3+4+5=31) attacked (5+3=8)

Not your turn.

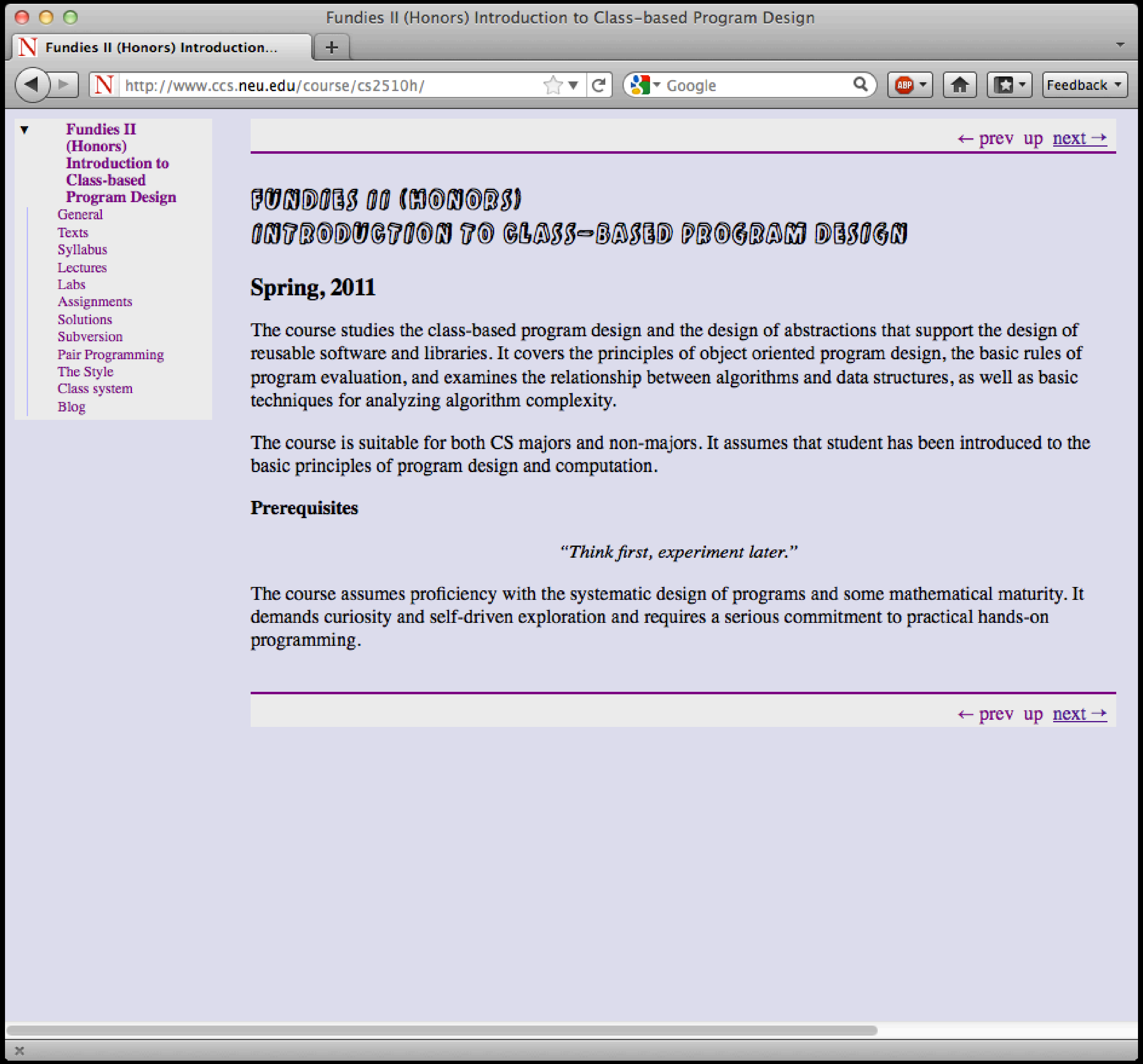
```
World signed up
(object:playing% '#hash((#<iworld> . (object:playe
-> World: (start 2 (((2 Player 2) (1 Player 1)) (((1 2
-> World: (start 1 (((2 Player 2) (1 Player 1)) (((1 2
-> World: turn
World ->: (name Rose-AI)
(object:playing% '#hash((#<iworld> . (object:player% #<iworld> "Player 2" 2)) (#<
-> World: error
World ->: (attack 2 0)
(object:playing% '#hash((#<iworld> . (object:player% #<iworld> "Player 2" 2)) (#<
-> World: (attack 2 (5 4 5 5 3 3 2 6 6) 0 (4 5 2 6 4) (((2 Player 2) (1 Player 1)) ((
-> World: (attack 2 (5 4 5 5 3 3 2 6 6) 0 (4 5 2 6 4) (((2 Player 2) (1 Player 1)) ((
World ->: done
```

The first year

- ★Inheritance
- ★Interfaces
- ★Distributed programming
- ★Delegation
- ★Abstraction
- ★Invariants
- ★Unit testing
- ★Random testing
- ★Types
- ★Mixins
- ★Overriding
- ★Visitors
- ★Mutation
- ★Equality
- ★Implementing OO
- ★Java
- ★Generics
- ★Ruby
- ★Artificial intelligence

...apologies to John Woo

The first year



Fundies II (Honors) Introduction to Class-based Program Design
Fundies II (Honors) Introduction...
http://www.ccs.neu.edu/course/cs2510h/
Google
Feedback

← prev up next →

Fundies II (Honors)
Introduction to Class-based Program Design

Spring, 2011

The course studies the class-based program design and the design of abstractions that support the design of reusable software and libraries. It covers the principles of object oriented program design, the basic rules of program evaluation, and examines the relationship between algorithms and data structures, as well as basic techniques for analyzing algorithm complexity.

The course is suitable for both CS majors and non-majors. It assumes that student has been introduced to the basic principles of program design and computation.

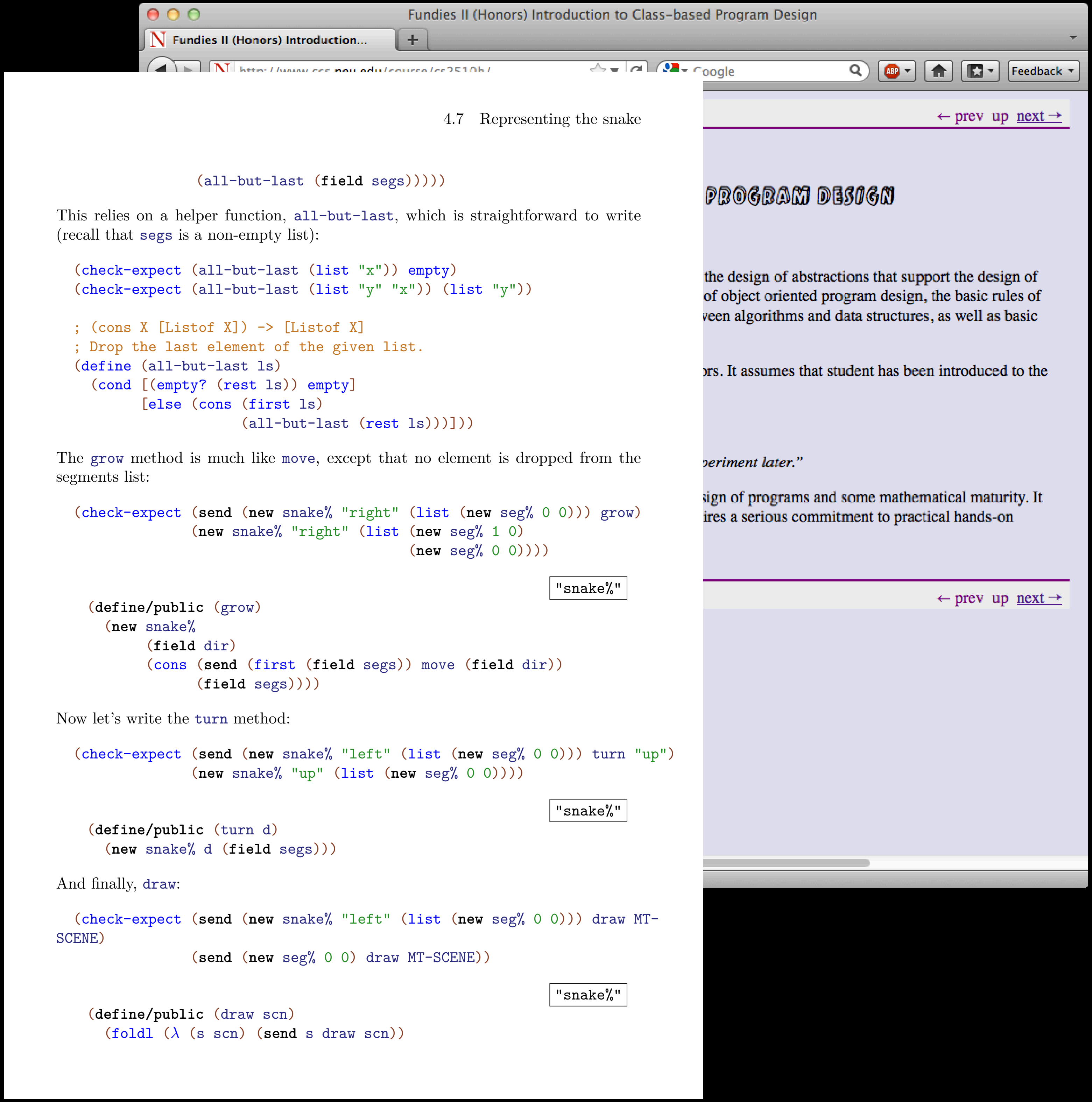
Prerequisites

"Think first, experiment later."

The course assumes proficiency with the systematic design of programs and some mathematical maturity. It demands curiosity and self-driven exploration and requires a serious commitment to practical hands-on programming.

← prev up next →

The first year


Fundies II (Honors) Introduction to Class-based Program Design
Fundies II (Honors) Introduction...
+
http://www.ccs.neu.edu/course/cs2510h/
Google
Feedback
← prev up next →
PROGRAM DESIGN
← prev up next →

the design of abstractions that support the design of
of object oriented program design, the basic rules of
veen algorithms and data structures, as well as basic

ors. It assumes that student has been introduced to the

periment later.”

sign of programs and some mathematical maturity. It
ires a serious commitment to practical hands-on

```
              (all-but-last (field segs)))))
```

This relies on a helper function, all-but-last, which is straightforward to write (recall that segs is a non-empty list):

```
  (check-expect (all-but-last (list "x")) empty)
  (check-expect (all-but-last (list "y" "x")) (list "y"))

  ; (cons X [Listof X]) -> [Listof X]
  ; Drop the last element of the given list.
  (define (all-but-last ls)
    (cond [(empty? (rest ls)) empty]
          [else (cons (first ls)
                      (all-but-last (rest ls)))]))
```

The grow method is much like move, except that no element is dropped from the segments list:

```
  (check-expect (send (new snake% "right" (list (new seg% 0 0))) grow)
                (new snake% "right" (list (new seg% 1 0)
                                          (new seg% 0 0))))

                                                  "snake%"
    (define/public (grow)
      (new snake%
           (field dir)
           (cons (send (first (field segs)) move (field dir))
                 (field segs))))
```

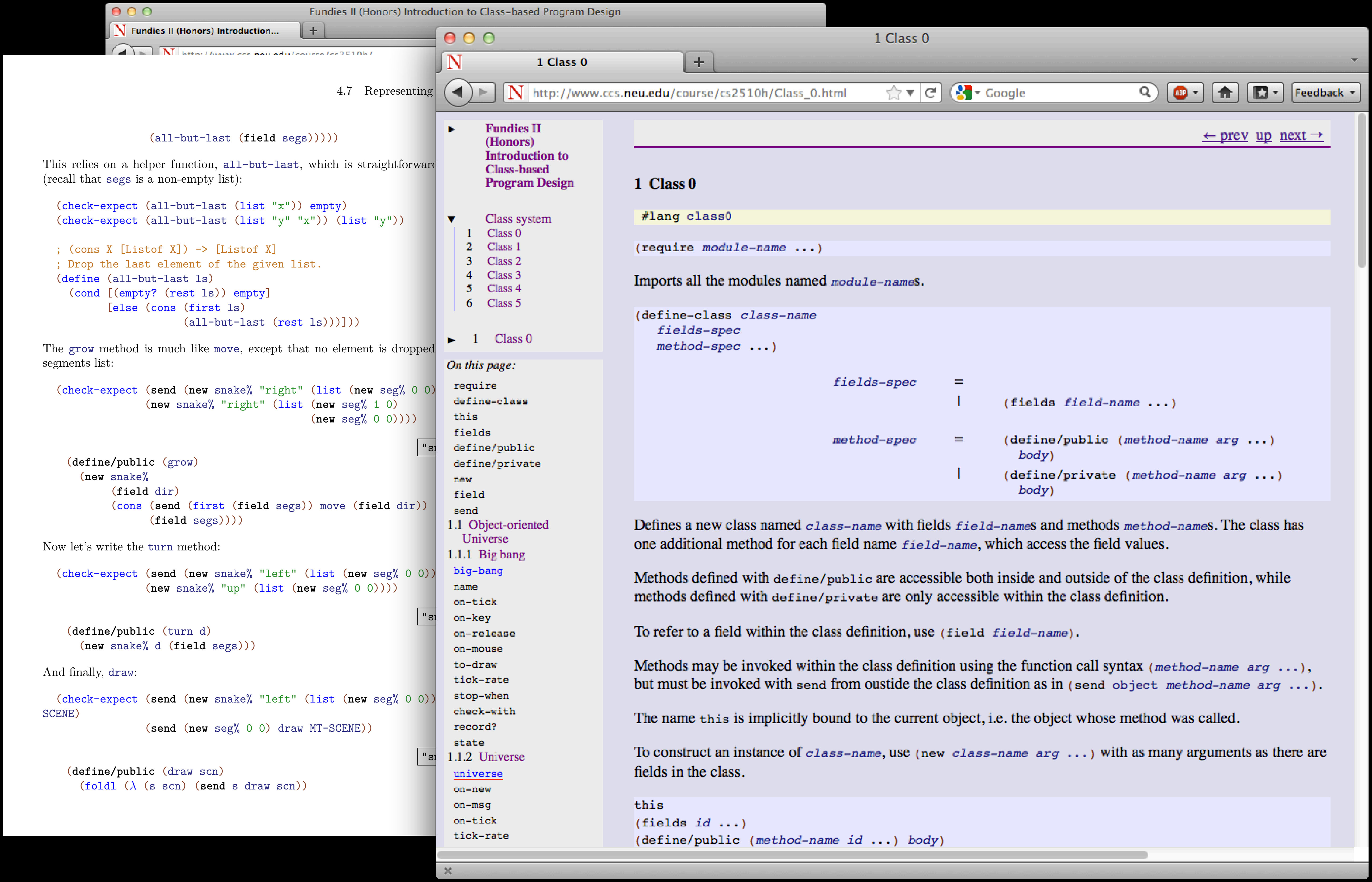
Now let’s write the turn method:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))) turn "up")
                (new snake% "up" (list (new seg% 0 0))))

                                                  "snake%"
    (define/public (turn d)
      (new snake% d (field segs)))
```

And finally, draw:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))) draw MT-
SCENE)
                (send (new seg% 0 0) draw MT-SCENE))

                                                  "snake%"
    (define/public (draw scn)
      (foldl (λ (s scn) (send s draw scn))
```

The first year


Fundies II (Honors) Introduction to Class-based Program Design
Fundies II (Honors) Introduction...
http://www.ccs.neu.edu/course/cs2510h/
1 Class 0
1 Class 0
http://www.ccs.neu.edu/course/cs2510h/Class_0.html
Google
Feedback

4.7 Representing

```
                (all-but-last (field segs)))))
```

This relies on a helper function, `all-but-last`, which is straightforward (recall that `segs` is a non-empty list):

```
  (check-expect (all-but-last (list "x")) empty)
  (check-expect (all-but-last (list "y" "x")) (list "y"))

  ; (cons X [Listof X]) -> [Listof X]
  ; Drop the last element of the given list.
  (define (all-but-last ls)
    (cond [(empty? (rest ls)) empty]
          [else (cons (first ls)
                      (all-but-last (rest ls)))]))
```

The `grow` method is much like `move`, except that no element is dropped segments list:

```
  (check-expect (send (new snake% "right" (list (new seg% 0 0)
                (new snake% "right" (list (new seg% 1 0)
                                          (new seg% 0 0))))

                                                              "s

    (define/public (grow)
      (new snake%
           (field dir)
           (cons (send (first (field segs)) move (field dir))
                 (field segs))))
```

Now let's write the `turn` method:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))
                (new snake% "up" (list (new seg% 0 0))))

                                                              "s

    (define/public (turn d)
      (new snake% d (field segs)))
```

And finally, `draw`:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))
SCENE)
                (send (new seg% 0 0) draw MT-SCENE))

                                                              "s

    (define/public (draw scn)
      (foldl (λ (s scn) (send s draw scn))
```

← prev up next →

1 Class 0

```
#lang class0
```

```
(require module-name ...)
```

Imports all the modules named `module-name`s.

```
(define-class class-name
   fields-spec
   method-spec ...)

      fields-spec  =
                   |  (fields field-name ...)

      method-spec  =  (define/public (method-name arg ...)
                        body)
                   |  (define/private (method-name arg ...)
                        body)
```

Defines a new class named `class-name` with fields `field-name`s and methods `method-name`s. The class has one additional method for each field name `field-name`, which access the field values.

Methods defined with `define/public` are accessible both inside and outside of the class definition, while methods defined with `define/private` are only accessible within the class definition.

To refer to a field within the class definition, use `(field field-name)`.

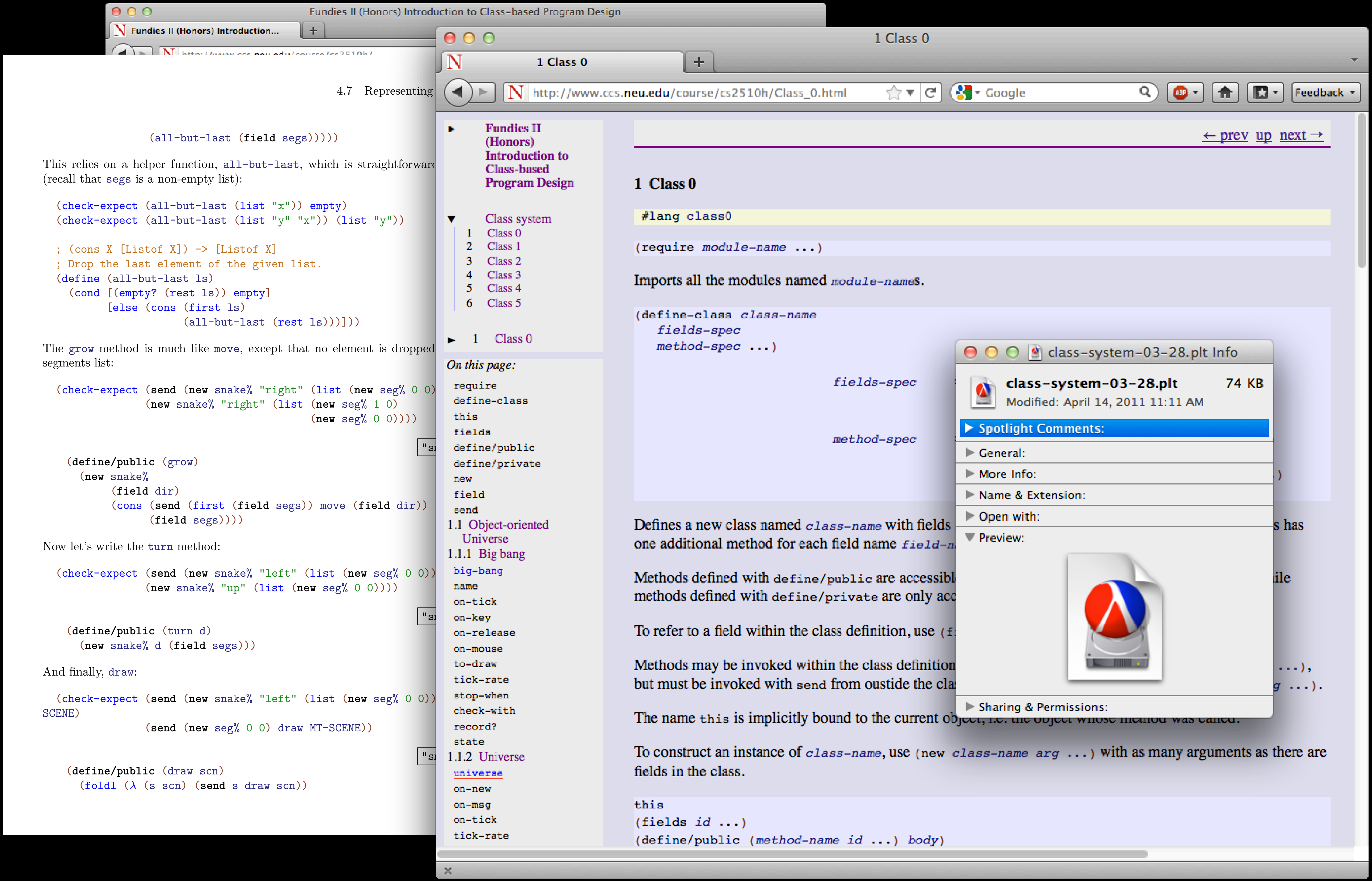
Methods may be invoked within the class definition using the function call syntax `(method-name arg ...)`, but must be invoked with `send` from oustide the class definition as in `(send object method-name arg ...)`.

The name `this` is implicitly bound to the current object, i.e. the object whose method was called.

To construct an instance of `class-name`, use `(new class-name arg ...)` with as many arguments as there are fields in the class.

```
this
(fields id ...)
(define/public (method-name id ...) body)
```

The first year



Fundies II (Honors) Introduction to Class-based Program Design

Fundies II (Honors) Introduction...

http://www.ccs.neu.edu/course/cs2510h/

4.7 Representing

```
        (all-but-last (field segs)))))
```

This relies on a helper function, all-but-last, which is straightforward (recall that segs is a non-empty list):

```
  (check-expect (all-but-last (list "x")) empty)
  (check-expect (all-but-last (list "y" "x")) (list "y"))

  ; (cons X [Listof X]) -> [Listof X]
  ; Drop the last element of the given list.
  (define (all-but-last ls)
    (cond [(empty? (rest ls)) empty]
          [else (cons (first ls)
                      (all-but-last (rest ls)))]))
```

The grow method is much like move, except that no element is dropped segments list:

```
  (check-expect (send (new snake% "right" (list (new seg% 0 0)
                (new snake% "right" (list (new seg% 1 0)
                                          (new seg% 0 0))))
                                                            "s

    (define/public (grow)
      (new snake%
           (field dir)
           (cons (send (first (field segs)) move (field dir))
                 (field segs))))
```

Now let's write the turn method:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))
                (new snake% "up" (list (new seg% 0 0))))
                                                            "s

    (define/public (turn d)
      (new snake% d (field segs)))
```

And finally, draw:

```
  (check-expect (send (new snake% "left" (list (new seg% 0 0))
SCENE)
                (send (new seg% 0 0) draw MT-SCENE))
                                                            "s

    (define/public (draw scn)
      (foldl (λ (s scn) (send s draw scn))
```

1 Class 0

1 Class 0

http://www.ccs.neu.edu/course/cs2510h/Class_0.html

1 Class 0

```
#lang class0
```

```
(require module-name ...)
```

Imports all the modules named *module-name*s.

```
(define-class class-name
   fields-spec
   method-spec ...)

                         fields-spec

                         method-spec
```

Defines a new class named *class-name* with fields … s has one additional method for each field name *field-n*…

Methods defined with define/public are accessibl… ile methods defined with define/private are only acc…

To refer to a field within the class definition, use (f…

Methods may be invoked within the class definition … ...), but must be invoked with send from oustide the cla… g ...).

The name this is implicitly bound to the current object, i.e. the object whose method was called.

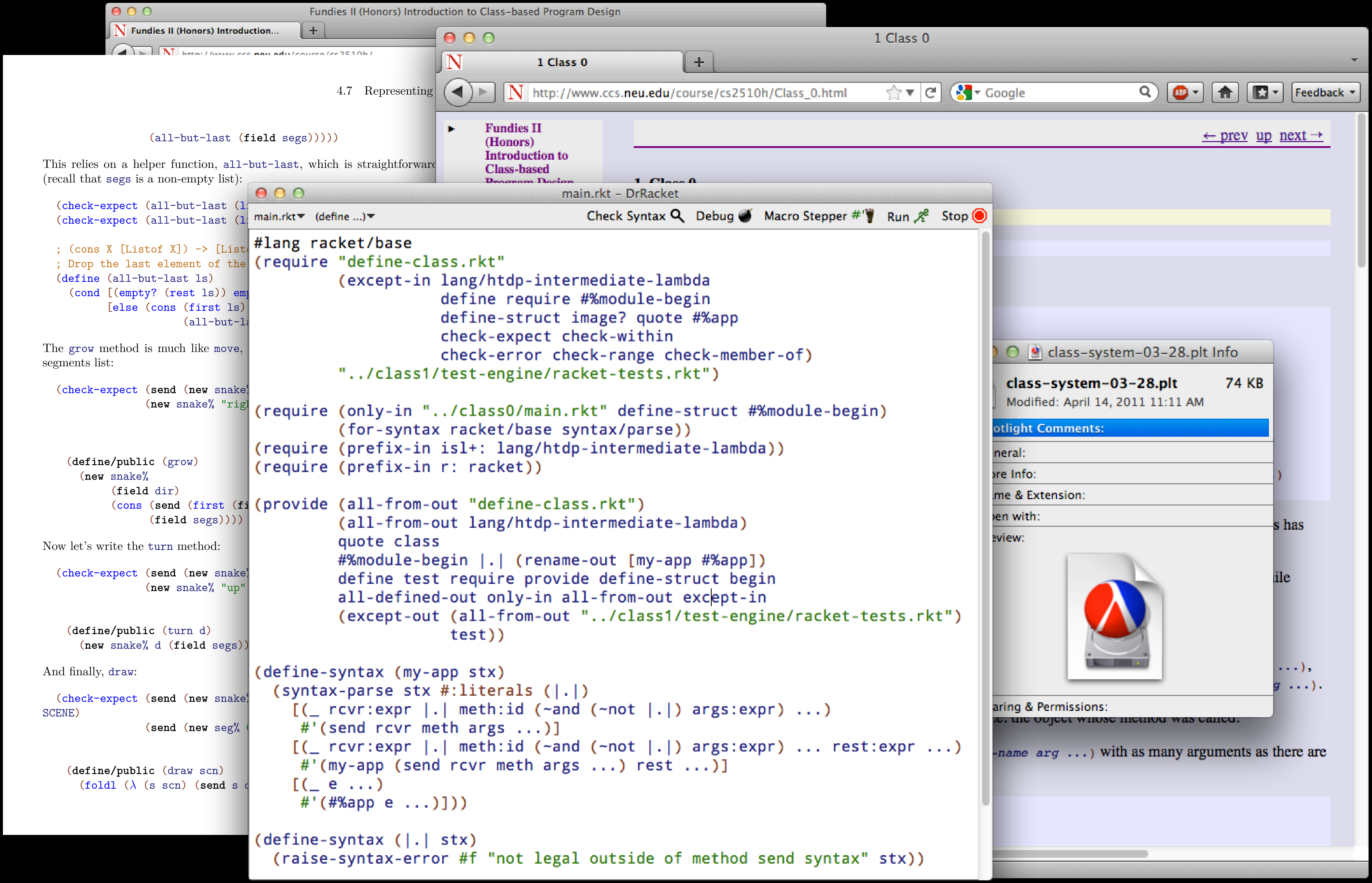
To construct an instance of *class-name*, use (new *class-name arg* ...) with as many arguments as there are fields in the class.

```
this
(fields id ...)
(define/public (method-name id ...) body)
```

class-system-03-28.plt Info

class-system-03-28.plt 74 KB
Modified: April 14, 2011 11:11 AM

The first year



Fundies II (Honors) Introduction to Class-based Program Design
Fundies II (Honors) Introduction...
http://www.ccs.neu.edu/course/cs2510h/
1 Class 0
1 Class 0
http://www.ccs.neu.edu/course/cs2510h/Class_0.html
Google
Feedback
Fundies II (Honors) Introduction to Class-based Program Design
← prev up next →
1 Class 0
main.rkt – DrRacket
main.rkt▾ (define ...)▾
Check Syntax
Debug
Macro Stepper
Run
Stop
class-system-03-28.plt Info
class-system-03-28.plt 74 KB
Modified: April 14, 2011 11:11 AM
otlight Comments:
neral:
ore Info:
me & Extension:
en with:
eview:
aring & Permissions:

4.7 Representing

```
        (all-but-last (field segs)))))
```

This relies on a helper function, all-but-last, which is straightforward
(recall that segs is a non-empty list):

```
  (check-expect (all-but-last (l
  (check-expect (all-but-last (l

  ; (cons X [Listof X]) -> [List
  ; Drop the last element of the
  (define (all-but-last ls)
    (cond [(empty? (rest ls)) em
          [else (cons (first ls)
                      (all-but-l
```

The grow method is much like move,
segments list:

```
  (check-expect (send (new snake
                (new snake% "rig

    (define/public (grow)
      (new snake%
           (field dir)
           (cons (send (first (fi
                 (field segs))))
```

Now let's write the turn method:

```
  (check-expect (send (new snake
                (new snake% "up"

    (define/public (turn d)
      (new snake% d (field segs))
```

And finally, draw:

```
  (check-expect (send (new snake
SCENE)
                (send (new seg% (

    (define/public (draw scn)
      (foldl (λ (s scn) (send s c
```

```
#lang racket/base
(require "define-class.rkt"
         (except-in lang/htdp-intermediate-lambda
                    define require #%module-begin
                    define-struct image? quote #%app
                    check-expect check-within
                    check-error check-range check-member-of)
         "../class1/test-engine/racket-tests.rkt")

(require (only-in "../class0/main.rkt" define-struct #%module-begin)
         (for-syntax racket/base syntax/parse))
(require (prefix-in isl+: lang/htdp-intermediate-lambda))
(require (prefix-in r: racket))

(provide (all-from-out "define-class.rkt")
         (all-from-out lang/htdp-intermediate-lambda)
         quote class
         #%module-begin |.| (rename-out [my-app #%app])
         define test require provide define-struct begin
         all-defined-out only-in all-from-out except-in
         (except-out (all-from-out "../class1/test-engine/racket-tests.rkt")
                     test))

(define-syntax (my-app stx)
  (syntax-parse stx #:literals (|.|)
    [(_ rcvr:expr |.| meth:id (~and (~not |.|) args:expr) ...)
     #'(send rcvr meth args ...)]
    [(_ rcvr:expr |.| meth:id (~and (~not |.|) args:expr) ... rest:expr ...)
     #'(my-app (send rcvr meth args ...) rest ...)]
    [(_ e ...)
     #'(#%app e ...)]))

(define-syntax (|.| stx)
  (raise-syntax-error #f "not legal outside of method send syntax" stx))
```

s has

ile

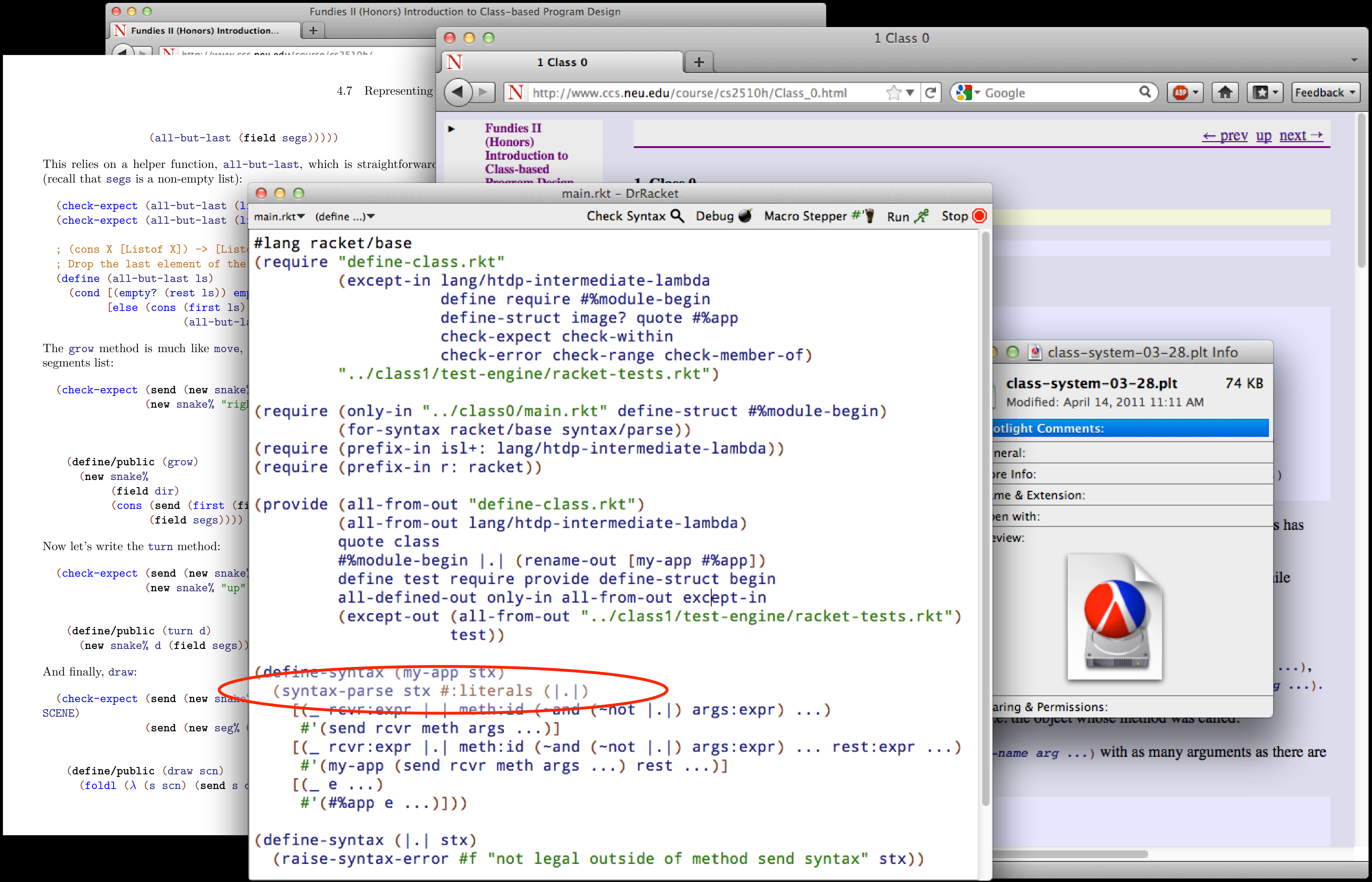
...),

g ...).

e. the object whose method was called.

-name arg ...) with as many arguments as there are

The first year



Fundies II (Honors) Introduction to Class-based Program Design
Fundies II (Honors) Introduction...
http://www.ccs.neu.edu/course/cs2510h/
1 Class 0
http://www.ccs.neu.edu/course/cs2510h/Class_0.html
Google
Feedback
Fundies II (Honors) Introduction to Class-based Program Design
← prev up next →
1 Class 0
main.rkt – DrRacket
main.rkt▾ (define ...)▾
Check Syntax
Debug
Macro Stepper #'
Run
Stop
class-system-03-28.plt Info
class-system-03-28.plt 74 KB
Modified: April 14, 2011 11:11 AM
otlight Comments:
neral:
ore Info:
me & Extension:
pen with:
eview:
aring & Permissions:

4.7 Representing

```
                    (all-but-last (field segs)))))
```

This relies on a helper function, all-but-last, which is straightforward
(recall that segs is a non-empty list):

```
  (check-expect (all-but-last (l
  (check-expect (all-but-last (l

  ; (cons X [Listof X]) -> [Listo
  ; Drop the last element of the
  (define (all-but-last ls)
    (cond [(empty? (rest ls)) em
          [else (cons (first ls)
                      (all-but-l
```

The grow method is much like move,
segments list:

```
  (check-expect (send (new snake
                (new snake% "rig

    (define/public (grow)
      (new snake%
           (field dir)
           (cons (send (first (fi
                 (field segs))))
```

Now let's write the turn method:

```
  (check-expect (send (new snake
                (new snake% "up"

    (define/public (turn d)
      (new snake% d (field segs))
```

And finally, draw:

```
  (check-expect (send (new snake
SCENE)
                (send (new seg% (

    (define/public (draw scn)
      (foldl (λ (s scn) (send s d
```

```
#lang racket/base
(require "define-class.rkt"
         (except-in lang/htdp-intermediate-lambda
                    define require #%module-begin
                    define-struct image? quote #%app
                    check-expect check-within
                    check-error check-range check-member-of)
         "../class1/test-engine/racket-tests.rkt")

(require (only-in "../class0/main.rkt" define-struct #%module-begin)
         (for-syntax racket/base syntax/parse))
(require (prefix-in isl+: lang/htdp-intermediate-lambda))
(require (prefix-in r: racket))

(provide (all-from-out "define-class.rkt")
         (all-from-out lang/htdp-intermediate-lambda)
         quote class
         #%module-begin |.| (rename-out [my-app #%app])
         define test require provide define-struct begin
         all-defined-out only-in all-from-out except-in
         (except-out (all-from-out "../class1/test-engine/racket-tests.rkt")
                     test))

(define-syntax (my-app stx)
  (syntax-parse stx #:literals (|.|)
    [(_ rcvr:expr |.| meth:id (~and (~not |.|) args:expr) ...)
     #'(send rcvr meth args ...)]
    [(_ rcvr:expr |.| meth:id (~and (~not |.|) args:expr) ... rest:expr ...)
     #'(my-app (send rcvr meth args ...) rest ...)]
    [(_ e ...)
     #'(#%app e ...)]))

(define-syntax (|.| stx)
  (raise-syntax-error #f "not legal outside of method send syntax" stx))
```

s has

ile

...),
g ...).

c. the object whose method was called.

-name arg ...) with as many arguments as there are

Magic Eight Ball

The next years

Bigger data designs

A good story for constructors

Better error messages

Types in class34

Whalesong?

Thanks!

`http://www.ccs.neu.edu/course/cs2510h/`

`{dvanhorn,samth}@ccs.neu.edu`