# CS 2010
# Computer Science I

Instructor: **Matthew Flatt**

## This Course is About...

Fundamentals of programming

- From specification to implementation
- Software engineering principles

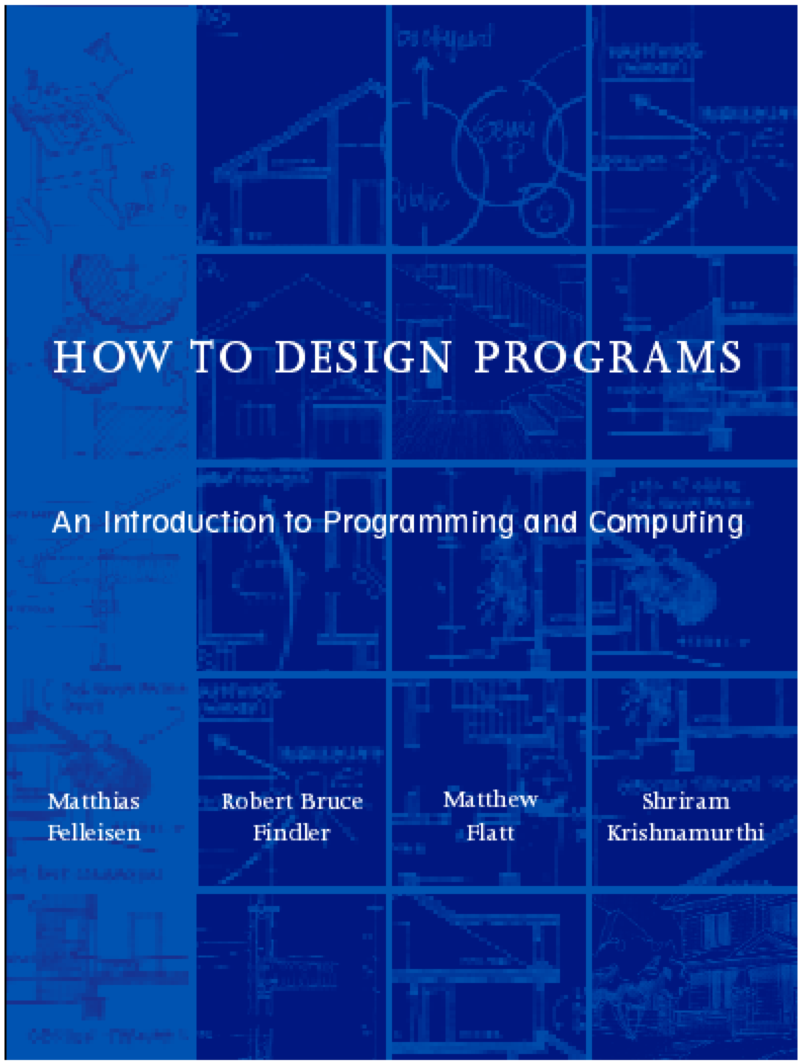
## This Course is...

***Not*** about...

- A particular programming language (e.g., Java, C++, Scheme)
- A particular programming tool (e.g., gcc, DrScheme)
- Specific libraries or protocols (e.g., Gtk, XML, HTTP)
- How programs get translated into electronic signals

## Book

*How to Design Programs*


HOW TO DESIGN PROGRAMS

An Introduction to Programming and Computing

Matthias Felleisen

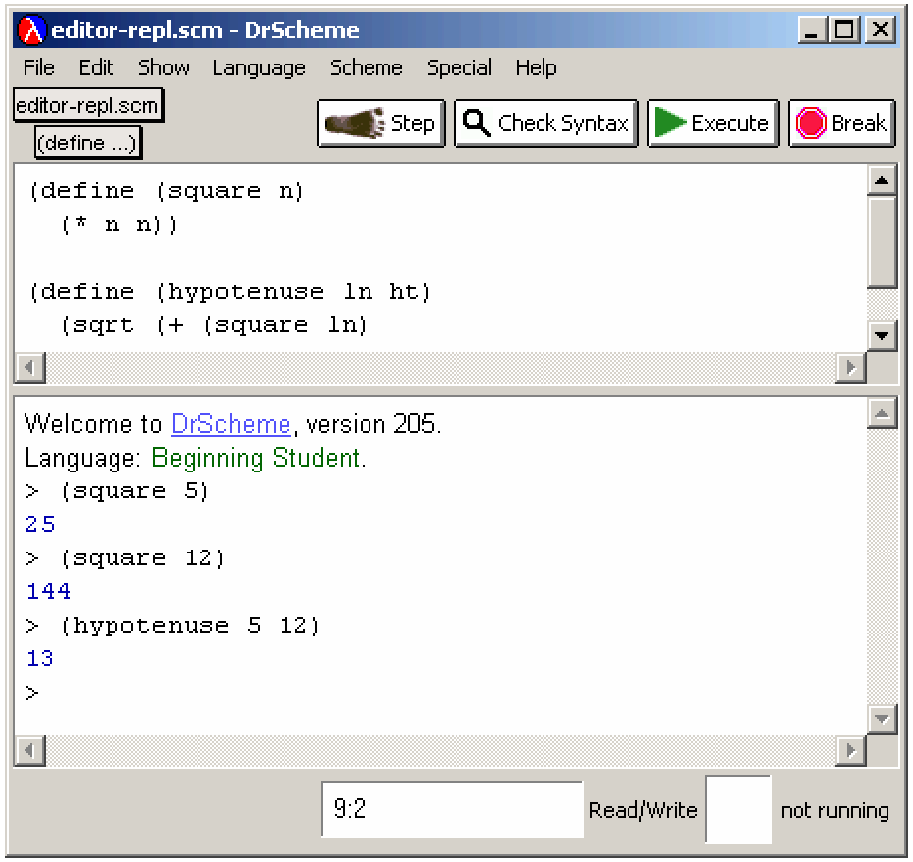
Robert Bruce Findler

Matthew Flatt

Shriram Krishnamurthi

# Programming Environment

*DrScheme*


editor-repl.scm - DrScheme
File Edit Show Language Scheme Special Help
editor-repl.scm
(define ...)
Step
Check Syntax
Execute
Break

```
(define (square n)
  (* n n))

(define (hypotenuse ln ht)
  (sqrt (+ (square ln)
```

```
Welcome to DrScheme, version 205.
Language: Beginning Student.
> (square 5)
25
> (square 12)
144
> (hypotenuse 5 12)
13
>
```

## What is Scheme?

- **Scheme** is a programming language
    - Used to implement DrScheme, for example
- The language for this course matches a subset of Scheme
- The course content is not Scheme-specific

## Getting Started:

## Arithmetic, Algebra, and Computing

## Arithmetic is Computing

- Fixed, pre-defined rules for ***primitive operators***:

$$2 + 3 = 5$$

$$4 \times 2 = 8$$

$$\cos(0) = 1$$

## Arithmetic is Computing

- Fixed, pre-defined rules for ***primitive operators***:

$$2 + 3 \rightarrow 5$$

$$4 \times 2 \rightarrow 8$$

$$\cos(0) \rightarrow 1$$

- Rules for combining other rules:
  - Evaluate sub-expressions first

$$4 \times (2 + 3) \rightarrow 4 \times 5 \rightarrow 20$$

  - Precedence determines subexpressions:

$$4 + 2 \times 3 \rightarrow 4 + 6 \rightarrow 10$$

## Algebra as Computing

- Definition:

$$f(x) = \cos(x) + 2$$

- Expression:

$$f(0) \;\rightarrow\; \cos(0) + 2 \;\rightarrow\; 1 + 2 \;\rightarrow\; 3$$

- First step uses the ***substitution*** rule for functions

## Notation

- Why do some primitive operators go in the middle, like +, while others go at the front, like cos?
- What are the precedence rules?
- How do we know which arguments go with which operators?
- Which parentheses are redundant?
- When does = mean definition and when does it mean a computation step?
- ...

## Simplified Expression Notation

- Put all operators at the front
- Start every operation with an open parenthesis
- Put a close parenthesis after the last argument
- Never add extra parentheses

| **Old** | **New** |
|---|---|
| $1 + 2$ | `(+ 1 2)` |
| $4 + 2 \times 3$ | `(+ 4 (* 2 3))` |
| $\cos(0) + 1$ | `(+ (cos 0) 1)` |

## Simplified Definition Notation

- Use the keyword **`define`** instead of =
- Put **`define`** at the front, and group with parentheses
- Move open parenthesis from after function name to before

| Old | New |
|---|---|
| f(x) = cos(x) + 2 | `(define (f x) (+ (cos x) 2))` |

- Move open parenthesis in function calls

| Old | New |
|---|---|
| f(0) | `(f 0)` |
| f(2+3) | `(f (+ 2 3))` |

## Evaluation is the Same as Before

```
(define (f x) (+ (cos x) 2))

(f 0)
```

## Evaluation is the Same as Before

```
(define (f x) (+ (cos x) 2))

(f 0)
 →  (+ (cos 0) 2)
```

## Evaluation is the Same as Before

```
(define (f x) (+ (cos x) 2))

(f 0)
 →  (+ (cos 0) 2)
 →  (+ 1 2)
```

## Evaluation is the Same as Before

```
(define (f x) (+ (cos x) 2))

(f 0)
 →  (+ (cos 0) 2)
 →  (+ 1 2)
 →  3
```

## Beyond Numbers: Booleans

Numbers are not the only kind of values:

| Old | | | New | | |
|---|---|---|---|---|---|
| $1 < 2$ | $\rightarrow$ | true | `(< 1 2)` | $\rightarrow$ | `true` |
| $1 > 2$ | $\rightarrow$ | true | `(> 1 2)` | $\rightarrow$ | `false` |
| $1 > 2$ | $\rightarrow$ | true | `(> 1 2)` | $\rightarrow$ | `false` |
| $2 \geq 2$ | $\rightarrow$ | true | `(>= 1 2)` | $\rightarrow$ | `true` |

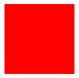
## Beyond Numbers: Booleans

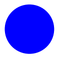
| Old | New |
|---|---|
| true and false | `(and true false)` |
| true or false | `(or true false)` |
| $1 < 2$ and $2 > 3$ | `(and (< 1 2) (> 2 3))` |
| $1 \leq 0$ and $1 = 1$ | `(or (<= 1 0) (= 1 1))` |
| $1 \neq 0$ | `(not (= 1 0))` |

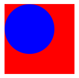
## Beyond Numbers: Symbols

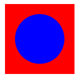
```
(symbol=? 'apple 'apple)  →  true
(symbol=? 'apple 'banana)  →  false
```

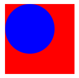
## Beyond Numbers: Images

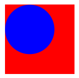
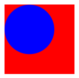
(filled-rect 35 35 'red) → 

(filled-circle 25 25 'blue) → 

(image+ ) → 

(offset-image+ 5 5 ) → 

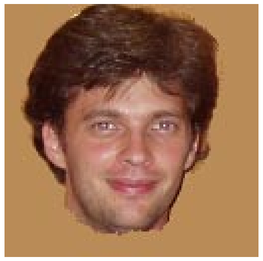
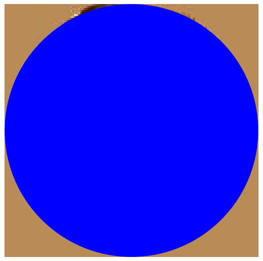
(image=? (image+ ) )

→ (image=?  )

→ true

## Programming with Images

```
(define (anonymize i)
  (offset-image+
   i
   0 0
   (filled-circle (image-width i)
                  (image-height i)
                  'blue)))
```

(anonymize  ) $\rightarrow$ ... $\rightarrow$ 

*Use the stepper to see all steps*