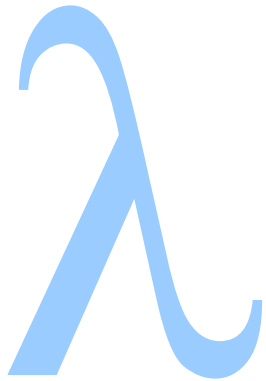


SCHEME
and functional programming

Dr. Jozo Dujmović

Programming paradigms

- Paradigm = fundamental style of programming.
- Main computer programming paradigms are:

1. Procedural (imperative) programming
2. Nonprocedural (declarative) programming
3. Logic programming (subset of declarative programming)
4. Functional programming (subset of declarative programming)
5. Object-oriented programming

Multi-paradigm languages

- Some languages strictly support only a single paradigm (Fortran, Cobol, Algol, Basic, PL/1, Pascal, C: mostly procedural)
- Some languages support multiple paradigms (C++ supports both procedural and object-oriented paradigm)

Imperative vs. Declarative

- Imperative programming is a programming paradigm that describes computation in terms of a program state (values of variables) and statements that change the program state. It is based on algorithms.
- Declarative programming specifies what the program should accomplish without describing control flow and algorithmic steps.

The concept of side effect

- A function or expression is supposed to return a value (and make no other effects).
- If, in addition to returning a value, a function modifies some state, it is said to produce a side effect.
- A goal of functional programming is to eliminate (or minimize) side effects
- Imperative programming uses side effects to produce program results.

Side effects and safety

- A “safe operation” is free of side effects
- Functions that use side effects may generate unexpected results and create unsafe programs.

```
int cube(int x)
{                          // g = a global variable
   g -= x;                 // Change of state = side effect
   return x*x*x;
}
```

cube(2)*cube(g) != cube(g)*cube(2) due to side effects

Mutable vs. immutable objects

- Mutable object is an object which can be modified after it is created (can change the state); e.g. int n;
- Immutable object is an object whose state cannot be modified after it is created (e.g. const double pi=3.14;)
- Advantages of immutable objects: no need to copy objects (which can be large), but only their reference (address, or pointer) – that improves performance. (Passing by value or reference)
- OO languages (e.g. Python, Ruby, Java) access objects using references.
- Some languages support both mutable and immutable objects

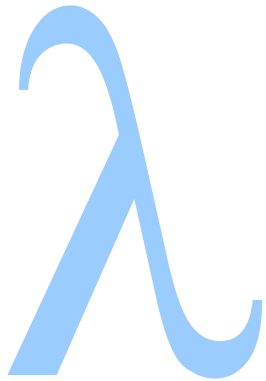
Functional programming

- One of fundamental programming paradigms
- Main properties:
  - Programming is organized as evaluation of functions
  - Avoiding mutable objects
  - Avoiding states, and changes of state
  - Avoiding side effects (function output depends only on arguments and not on state: f(X) always returns the same value for same X)
- Theoretical roots of functional programming are in lambda calculus

Lambda calculus

- Formal system that investigates function definitions, applications and recursion
- Can be interpreted as a minimalistic programming language capable of expressing any algorithm
- Uses unnamed (anonymous) functions without side effects
- Functions can accept as arguments and return as results other functions

LISP



- LISt Processing language
- Designed by John McCarthy
- First implemented by Steve Russell on an IBM 704 computer (first Lisp interpreter)
- The second oldest language in wide use today (FORTRAN [1957], LISP [1958])
- Favorite language for AI research
- Based on lambda calculus (Alonzo Church)
- Introduced many CS concepts (dynamic typing, tree data structures, recursive functions, OO programming, etc.)

LISP is a family of languages

Two main dialects of Lisp:

- Scheme
- Common Lisp

- There are many implementations of both Scheme and Common Lisp
- Both Scheme and Common Lisp have language standards

Main characteristics of LISP

- Main data structures are linked lists
- LISP source code is made of linked lists
- Recognizable fully parenthesized syntax
- Interchangeability of programs and data
- LISP programs can manipulate source code (can create new syntax and specific new languages embedded in LISP)

List notation

- Program statement:
  (function arg1 arg2 … argN)
- List containing data (“quoted list”):
  ‘(data1 data2 … dataN)
- Head and tail extraction functions have archaic names inherited from first two implementations of Lisp for the IBM 704:
  - Head: **car** (contents of the address part of register)
  - Tail: **cdr** (contents of the decrement part of register)

Car and cdr

- (car ‘(1 2 3 4))
- 1
- (cdr ‘(1 2 3 4))
- (2 3 4)

Anonymous functions

- The following function is anonymous:
  (lambda(n) (* n n))
- A general form of applying an operator:
  (<operator> <operand> … <operand>)
- Applying an anonymous function to a single argument
  > ((lambda(n) (* n n)) 5)
  > 25 (this is the returned value)

Scheme

- Scheme is a multi-paradigm language but the most important is its support of functional programming
- Developed by Guy L. Steele and Gerald Jay Sussman in the 1970s (1975)
- Successfully used in industry and commercial applications
- There are language standards (important for industrial software development)

First-class objects

- An object (data object, or a function) is first-class if it can be used in programs without restrictions:
  - May be expressed as an anonymous literal value (constant)
  - May be stored in variables
  - May be stored in data structures
  - May be comparable to other objects for equality
  - May be passed as parameter to procedures/functions
  - May be returned as result from procedures/functions
  - May be constructed at run time
  - Is readable and printable
  - It has intrinsic identity (independent of given name)
  - May be stored outside running processes
  - May be transmitted among processes
- In Scheme all objects (including functions) are first-class. Elimination of restrictions contributes to the expressive power of languages.

What Scheme authors think about programming languages

- “Programming languages should be designed not by **piling feature on top of feature**, but by **removing the weaknesses and restrictions** that make additional features appear necessary.”
- Therefore, the main goals are: simplicity, generality, and small number of powerful fundamental concepts.

Scheme Standards

- Mostly based on the latest version of the "Revised Report on the Algorithmic Language Scheme"
- IEEE Scheme – IEEE standard, 1178–1990 (R1995)

Scheme implementations

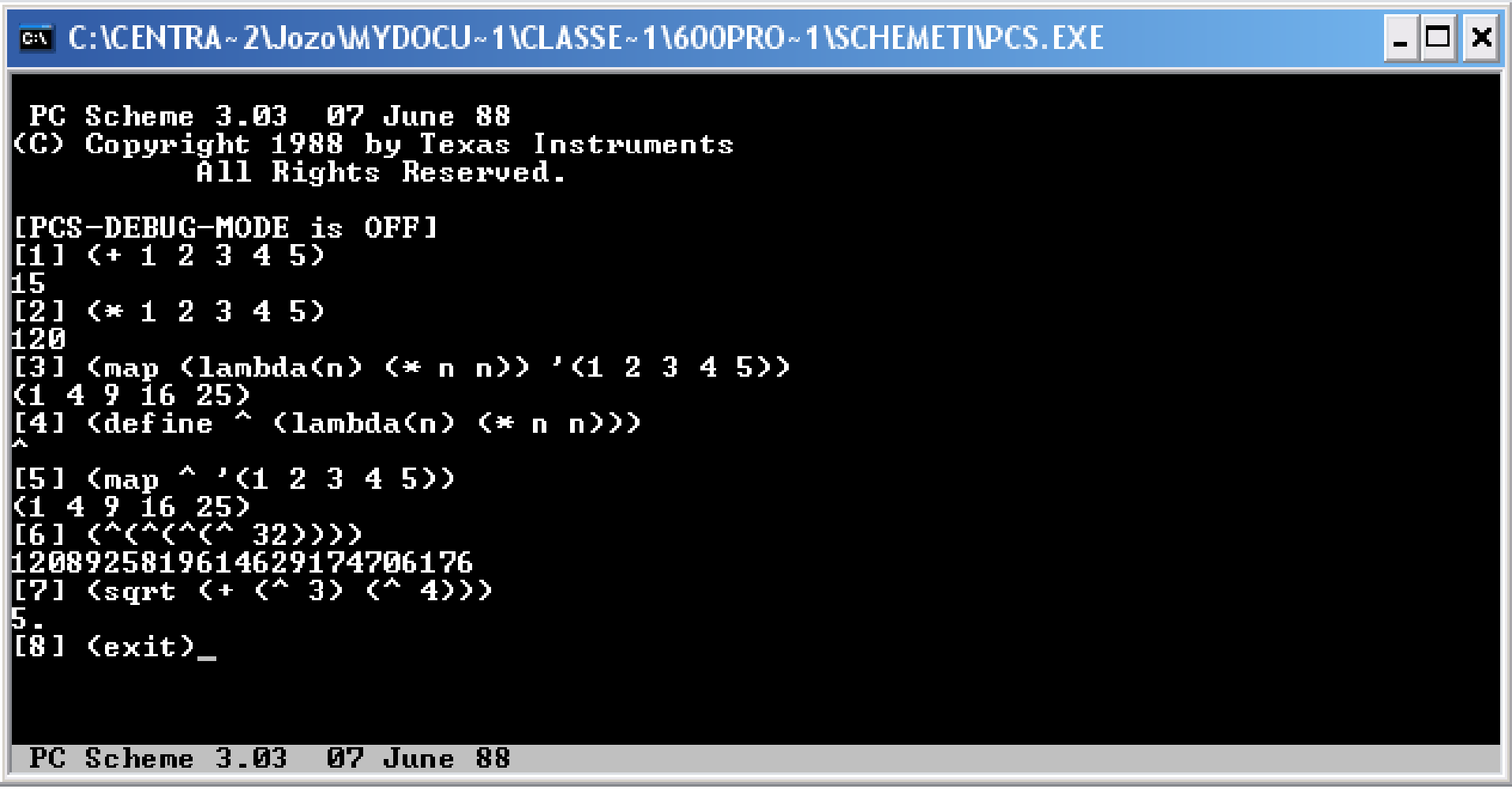
- Multiple free implementations are available on Internet
- MIT – GNU
- Texas Instruments
- University of Geneva Scheme
- Dr. Scheme

TI Scheme

- Can be installed by copying PCS.ZIP directory posted on iLearn
- Convenient for testing concepts and small programs
- Used in the Scheme part of the CSc 600 Reader
- Minimalistic features – just a DOS window
- Similar to the University of Geneva version

Texas Instruments Scheme

To start click PCS.EXE



DrScheme

- Comfortable environment based on GUI
- Related to a free on-line book **How to Design Programs**
- Two windows: function definition window and function execution window
- A version with traditional query-based interpreter (MzScheme)
- SW distribution: http://planet.plt-scheme.org/

How to Design Programs
An Introduction to Computing and Programming

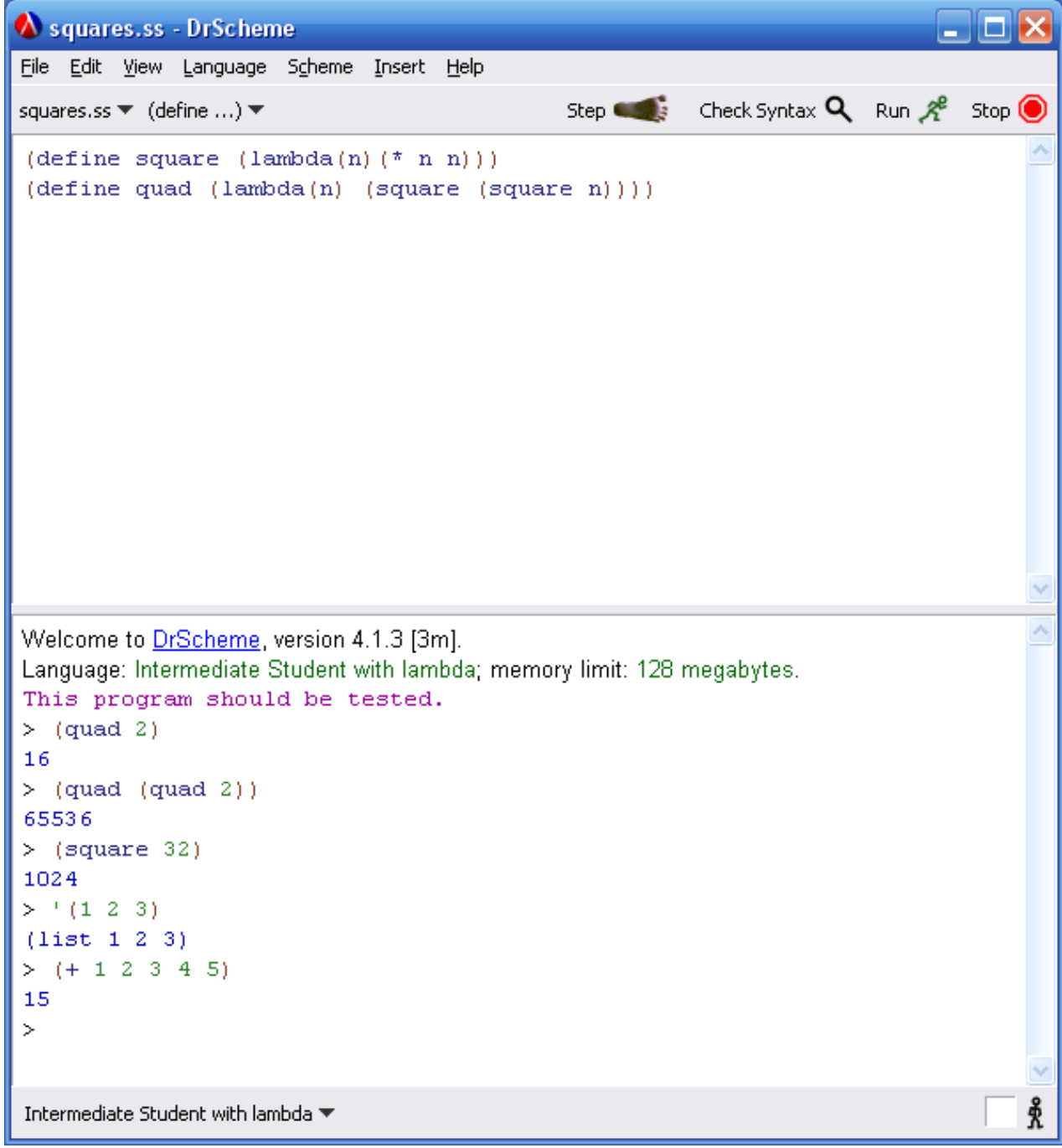
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi

Free Scheme book

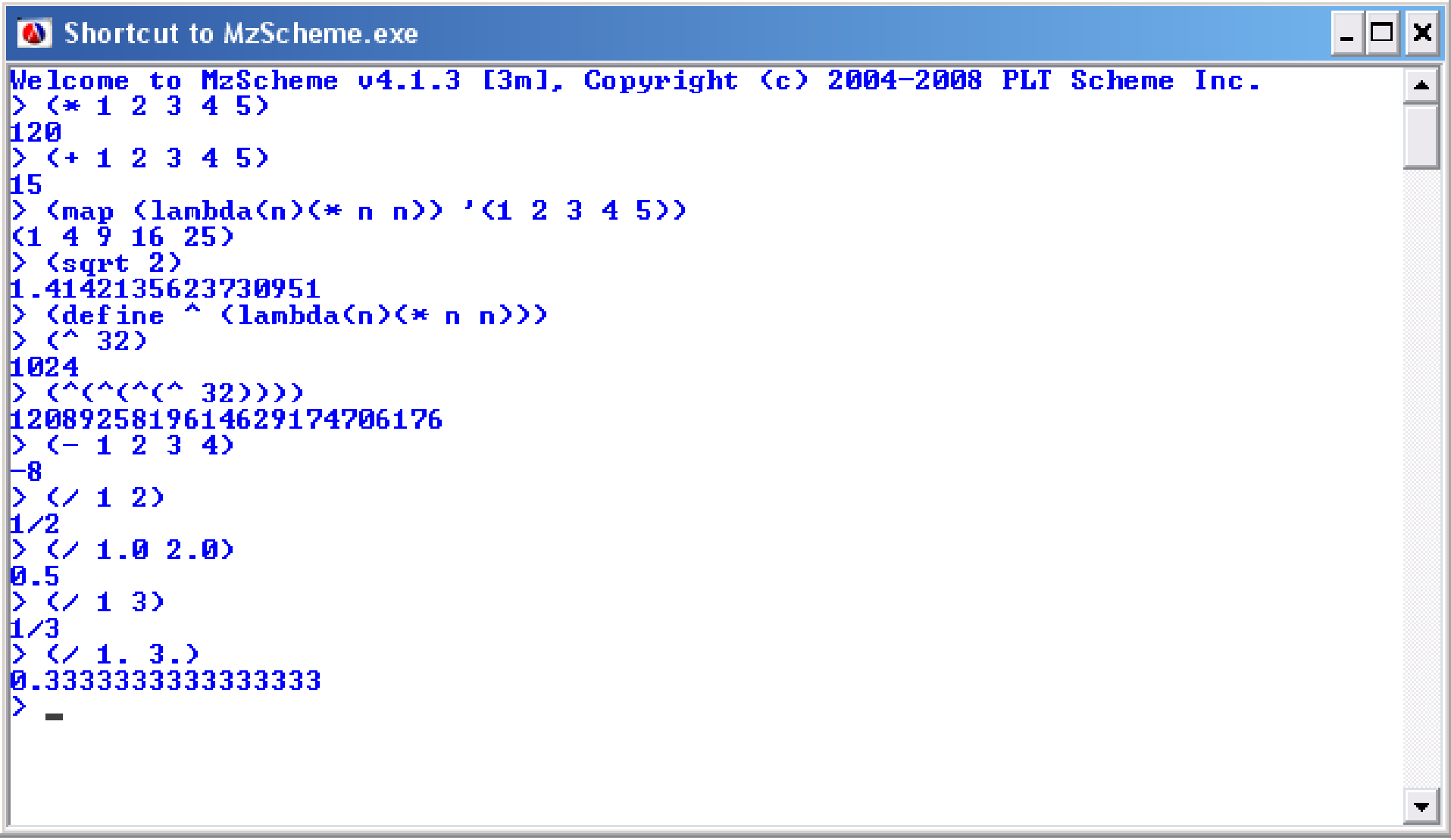
The MIT Press
Cambridge, Massachusetts
London, England
2001

Window for defining programs

Window for execution of programs



MzScheme


Shortcut to MzScheme.exe
Welcome to MzScheme v4.1.3 [3m], Copyright (c) 2004-2008 PLT Scheme Inc.
> (* 1 2 3 4 5)
120
> (+ 1 2 3 4 5)
15
> (map (lambda(n)(* n n)) '(1 2 3 4 5))
(1 4 9 16 25)
> (sqrt 2)
1.4142135623730951
> (define ^ (lambda(n)(* n n)))
> (^ 32)
1024
> (^(^(^(^ 32))))
1208925819614629174706176
> (- 1 2 3 4)
-8
> (/ 1 2)
1/2
> (/ 1.0 2.0)
0.5
> (/ 1 3)
1/3
> (/ 1. 3.)
0.3333333333333333
>