Introduction

CSE 413, Autumn 2002

Programming Languages

http://www.cs.washington.edu/education/courses/413/02au/

Readings and References

- Reading
  - » Sections 1-1.1.5, *Structure and Interpretation of Computer Programs*, by Abelson, Sussman, and Sussman
- Other References
  - » Everything related to the class is available from the class web site

    http://www.cs.washington.edu/education/courses/413/02au/
  - » Section 2, *Revised^5 Report on the Algorithmic Language Scheme (R5RS)*

Elements of Programming

- Primitive expressions
  - » simplest entities of the language
- Means of combination
  - » by which compound elements are built
- Means of abstraction
  - » by which compound elements can be named and manipulated as units

There are many "languages"

- Computer programming
  - » Basic, Cobol, C, Pascal, Ada, Java, Python, …
- Shell and scripting languages
  - » Perl, bash, AppleScript, JavaScript, ...
- Applications
  - » Photoshop, MS Office, Matlab, POVRay, ...
- Sciences
  - » DNA, Chemistry, Plant Growth, ...

Training and Education

- Training
  - » learn the specifics of a known language
  - » build up a "tool chest" so that you can perform specific tasks in a particular field
- Education
  - » learn how to recognize valid abstractions and synthesize them in new and useful ways in many different knowledge domains
- We'll do some of both in this class

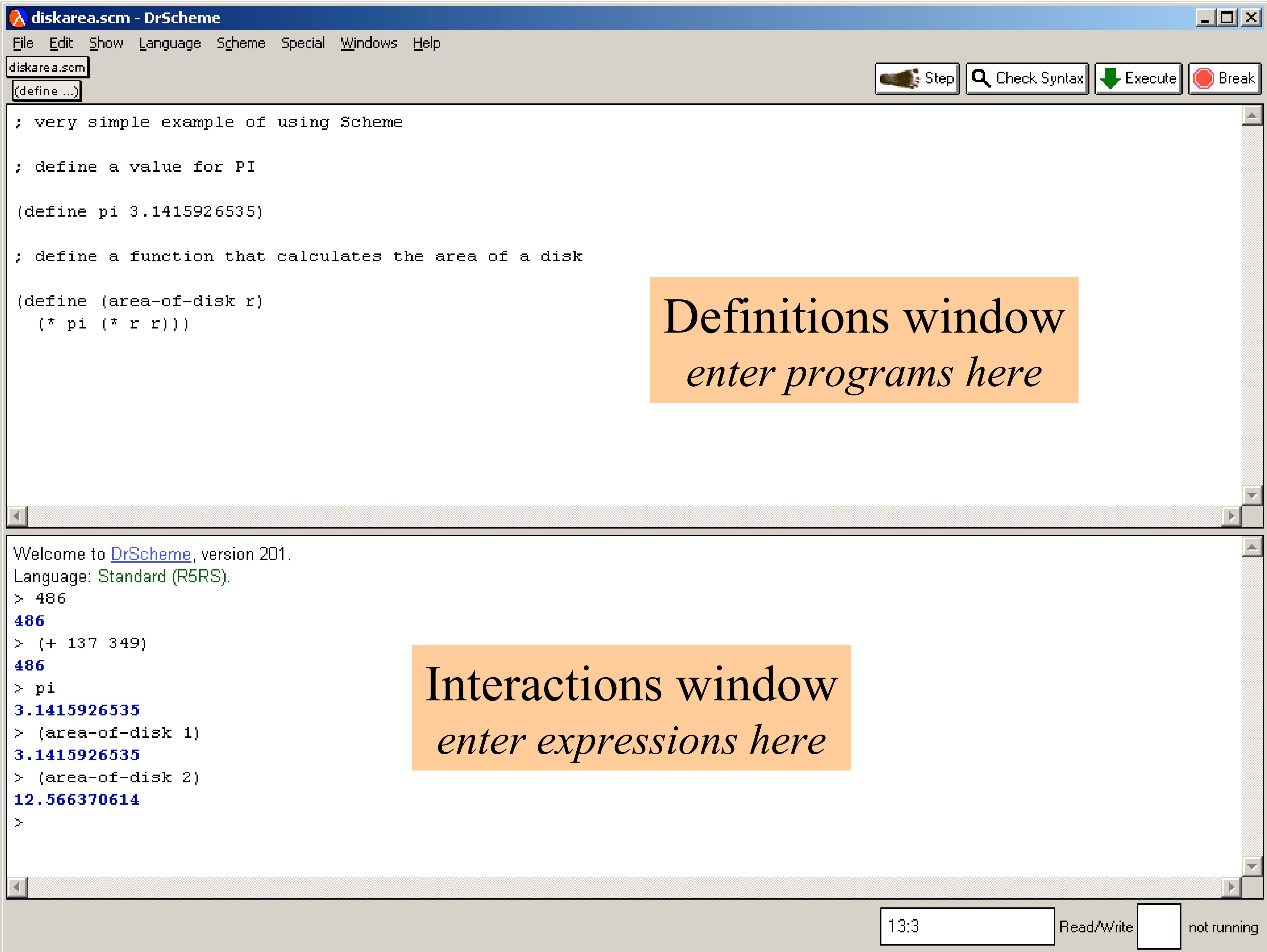
What is Scheme?

- Is Scheme a version of Lisp?
  - » Yes: Scheme has a strong syntactic resemblance to Lisp. Editing Scheme on a computer is much easier than editing most other syntaxes. Students take about one day to learn the syntax, and can then move on to learning real concepts.
  - » No: Beyond this, Scheme shares very little with Lisp. Don't be mislead by the syntactic similarity; Scheme is a fairly different language with a much more refined and modern philosophy.

http://www.teach-scheme.org/Notes/scheme-faq.html

Why Scheme?

- The simplicity of the language lets us work on problem solving, rather than just syntax issues
- Flexibility of the language lets us see that the structure of C/Java/Basic is not the only way to express problem solutions
- Variety is the spice of life
  - » study more than one language paradigm and study the relationship between design paradigms
  - » professional programmers switch languages every few years anyway, so start practicing now



diskarea.scm - DrScheme
File Edit Show Language Scheme Special Windows Help
diskarea.scm
(define ...)
Step
Check Syntax
Execute
Break
; very simple example of using Scheme
; define a value for PI
(define pi 3.1415926535)
; define a function that calculates the area of a disk
(define (area-of-disk r)
(* pi (* r r)))
Definitions window
enter programs here
Welcome to DrScheme, version 201.
Language: Standard (R5RS).
> 486
486
> (+ 137 349)
486
> pi
3.1415926535
> (area-of-disk 1)
3.1415926535
> (area-of-disk 2)
12.566370614
>
Interactions window
enter expressions here
13:3
Read/Write
not running

Definitions window

- Define programs in the Definitions window
  - » save the contents of the window to a file using menu item File - Save Definitions As …
  - » load existing files with menu item File - Open
  - » execute the contents of the definitions window by clicking on the "Execute" button
  - » check and highlight syntax by clicking on the "Check Syntax" button

Interactions Window

- Evaluate simple expressions directly in the Interactions window
- Position the cursor after the ">", then type in your expression
  - » DrScheme responds by evaluating the expression and printing the result
- Expressions can reference symbols defined when you executed the Definitions window

Think functionally

- Programming that makes extensive use of assignment is known as *imperative programming*
  - » The order of assignments changes the operation of the program because the state is changed by assignment
- Programming without the use of assignment statements is known as *functional programming*
  - » In such a language, all procedures implement well-defined mathematical functions of their arguments whose behavior does not change
  - » Scheme is heavily oriented towards *functional* style

Primitive Expressions

- constants
  - » integer : -1, 0 3
  - » rational : ½, ¾
  - » real : 0.333, 3.1415926535
  - » boolean : #t, #f
- variable names (symbols)
  - » Names can contain almost any character except white space and parentheses
  - » Stick with simple names like **`value, x, iter, ...`**

Compound Expressions

- Either a *combination* or a *special form*
- Combination : (operator operand operand …)
  - » there are quite a few pre-defined operators
    - +, *, abs, sin, etc
  - » We can define our own operators
    - area-of-disk
- Special form
  - » keywords in the language
  - » eg, define

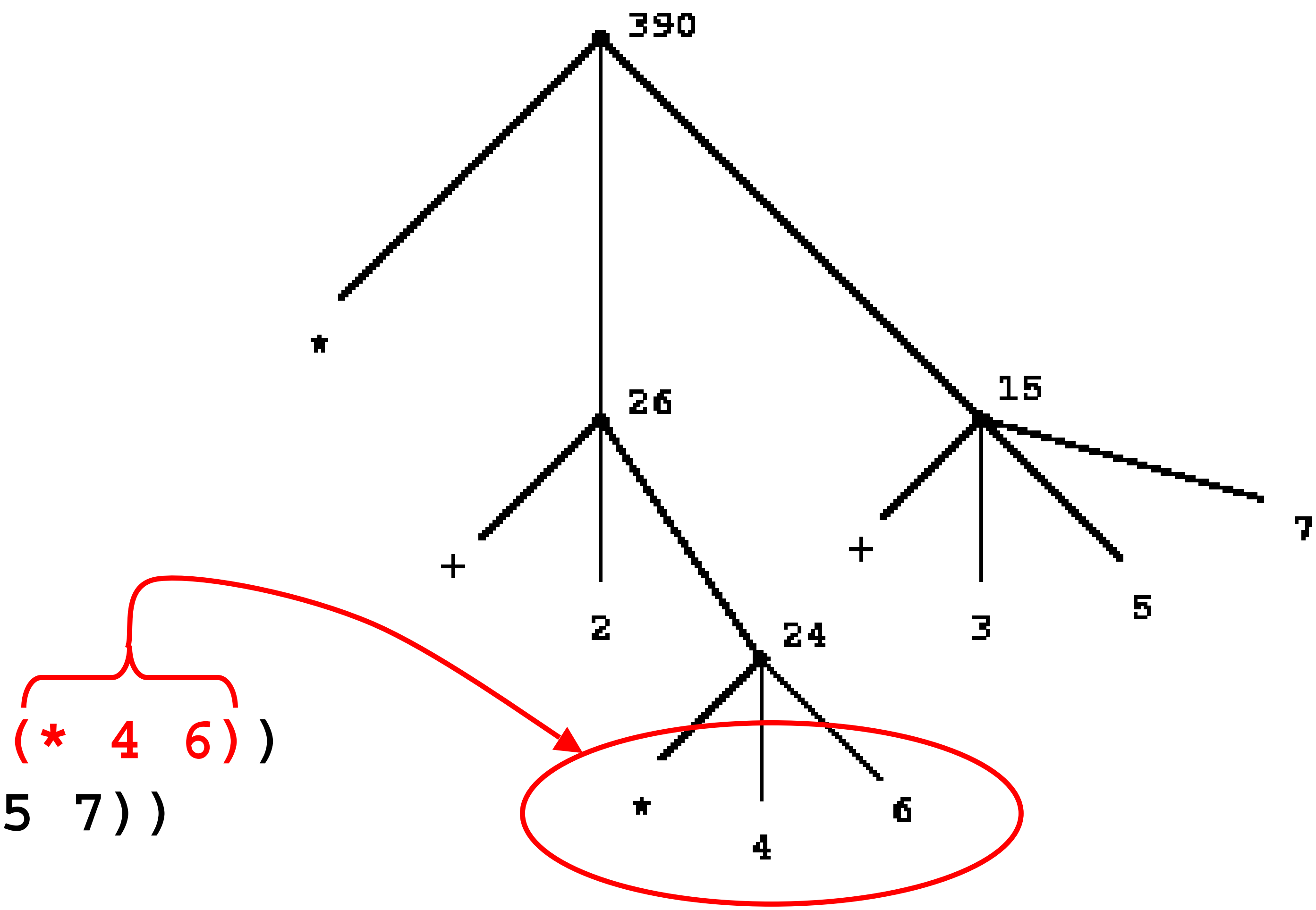
Combinations

- (operator operand operand …)
- this is prefix notation, the operator comes first
- a combination always denotes a procedure application
- the operator is a symbol or an expression, the applied procedure is the associated value
  - » +, -, abs, my-function, foop?
  - » characters like * and + are not special; if they do not stand alone then they are part of some name

Evaluating Combinations

- To evaluate a combination
  - » Evaluate the subexpressions of the combination
  - » Apply the procedure that is the value of the leftmost subexpression (the operator) to the arguments that are the values of the other subexpresions (the operands)
- For example
  - » `(* 5 99)` is a combination consisting of three subexpressions
  - » Scheme evaluates this combination and returns 495

Percolate values up a tree



Evaluate

```
(* (+ 2 (* 4 6))
   (+ 3 5 7))
```

Evaluating Special Forms

- Special forms have unique evaluation rules
- `(define x 3)` is an example of a special form; it is not a combination
  - » the evaluation rule for a simple define is "associate the given name with the given value"
- There are more special forms which we will encounter, but there are surprisingly few of them compared to other languages