Lecture 1: Basic Scheme

Welcome!

- Our Goal: Help you get you ready for 6.001
- Administrivia:
  - I'll be sending the class list to the registrar by Thursday.
  - We have a waiting list of students who would like to join the class
  - If you think you will drop this course, please decide soon and let me know ASAP

Homework

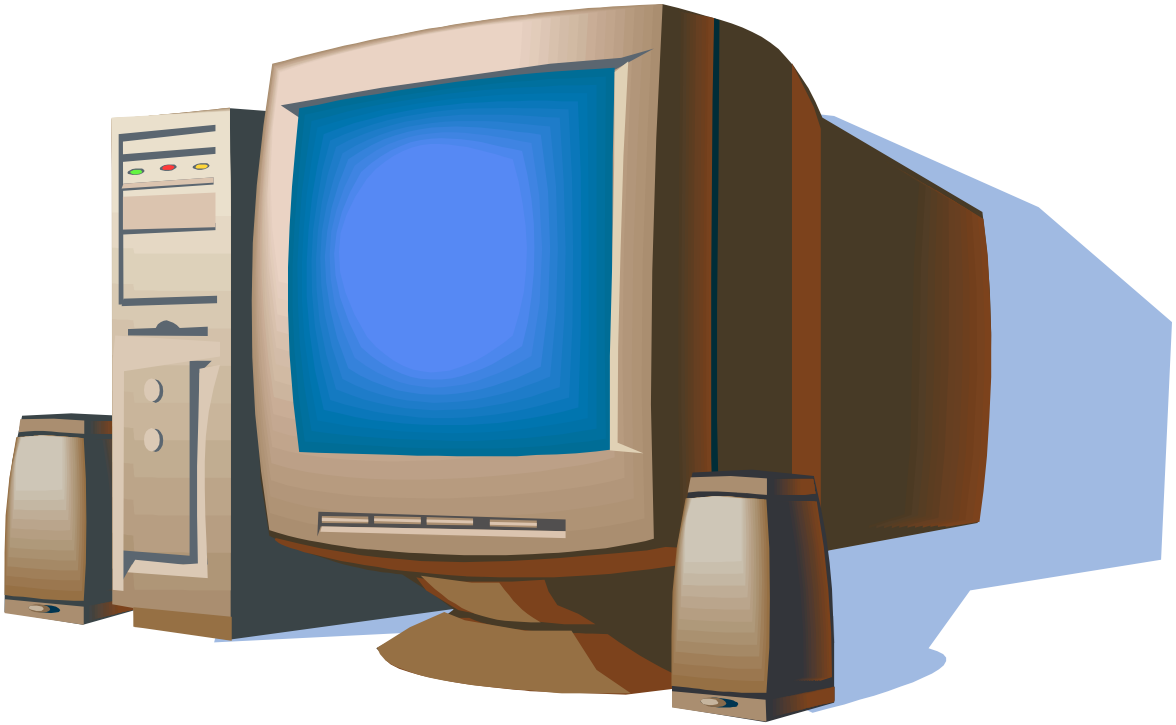
- From the course policy:
  - There will be seven homework assignments, you must do them all to pass the course
  - “Homework will be not be graded; any commentary is to point out things that you did well or could do better. In order to have "done" a homework, you must have put a significant amount of effort into completing it; all the assigned problems need not be working.”

Homework (cont.)

- Some of you may have discovered that this course was also taught last year and is in OpenCourseWare
- Some of the problem sets will be similar
- *Please* do not use these to complete your assignments.

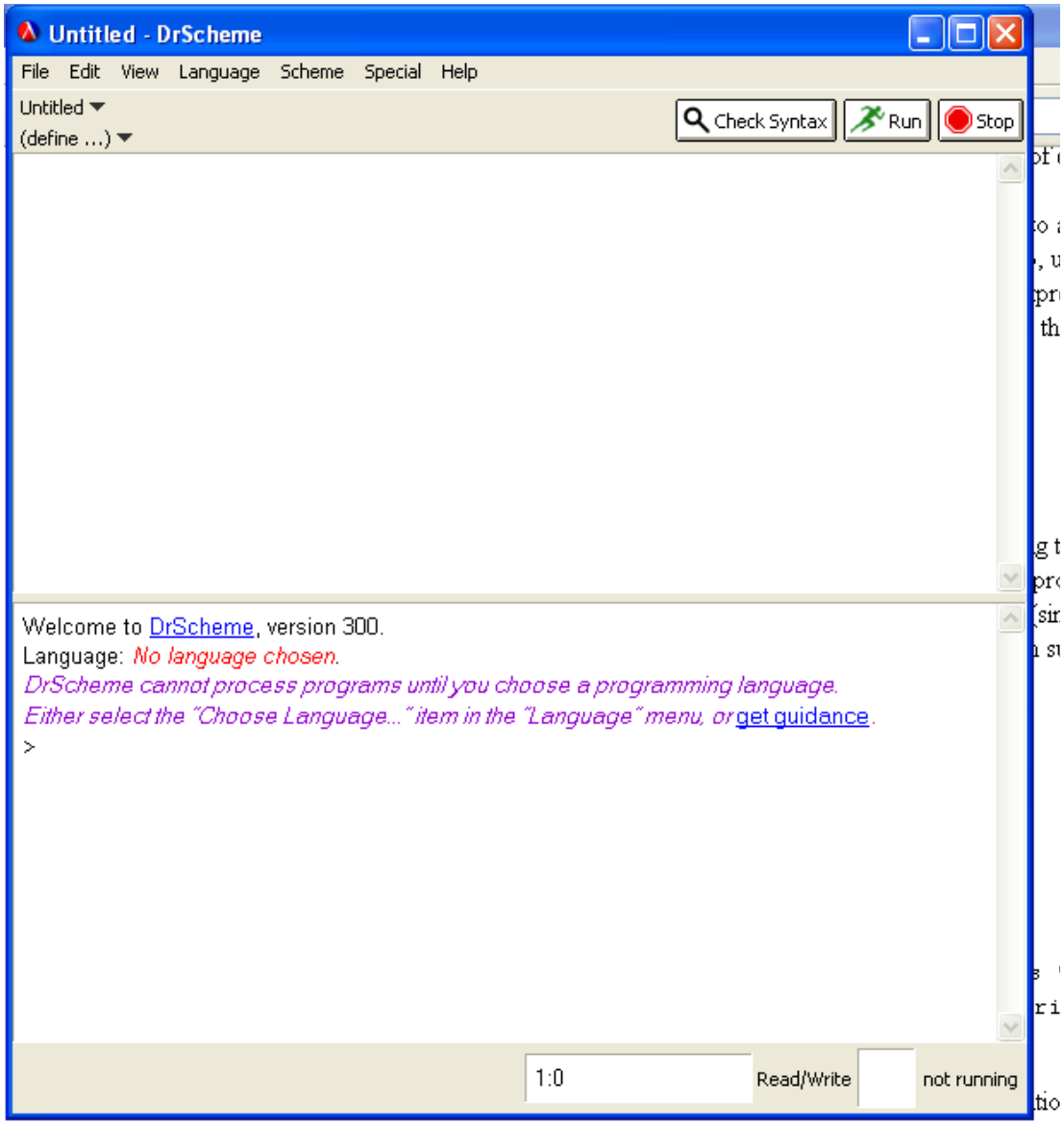
Laptops

Our Goal



Make this machine do what we want it to do

Our tool


Untitled - DrScheme
File Edit View Language Scheme Special Help
Untitled
(define ...)
Check Syntax
Run
Stop
Welcome to DrScheme, version 300.
Language: No language chosen.
DrScheme cannot process programs until you choose a programming language.
Either select the "Choose Language..." item in the "Language" menu, or get guidance.
>
1:0
Read/Write
not running

Scheme

- Scheme is a programming language that we learn to use to describe computational process
- Today we are going to talk about the basics of the language

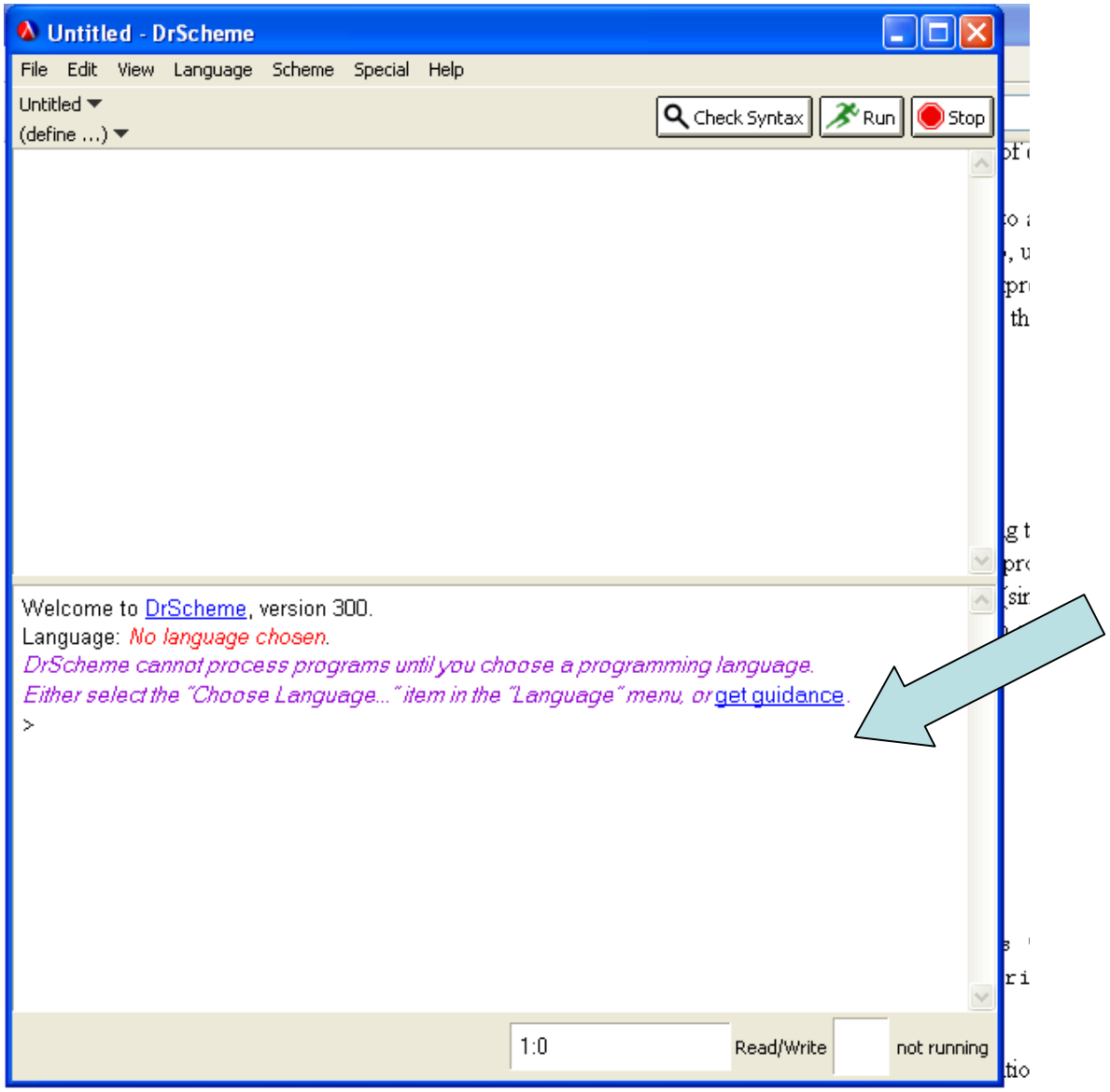
Scheme

- To make the computer do anything useful, we need 3 things
  - A way to represent data
  - Basic operations that we can perform on this data
  - Means to combine these operations to do more complicated things

Let's Do Some Computation

- We really want to compute 3 + 5
- In Scheme, most computation is described by *expressions* that are *evaluated*
- "Evaluate the expression 3 + 5" means "Perform the computation to add 3 and 5 then *return* the *value* of the expression"

Evaluating Expressions in Dr. Scheme



- Type expressions into this window
- Scheme will evaluate the expression, then display the value of the expression

The simplest expression

- Type 3 into the bottom window
- What happens?

The simplest expression

- Type 3 into the bottom window
- What happens?

- Scheme has
  - Read the expression that you typed in
  - Evaluated it
  - Then printed the answer

3 is an expression?

- A number is a type of *self-evaluating primitive*
- The value of the expression is the same as the expression
- Examples
  - Numbers – 1, 2, 3, 3.2, 4
  - Strings – “Go Sox!”
  - Booleans - #t, #f
  - Procedures

Types

- It is important to remember that every value has a type.
- Some operations are only defined for certain types
- “Go Sox!” + “Yankees stink!” is not defined in Scheme

Procedures are a type?

- A primitive procedure is a built-in procedure to manipulate objects
  - Numbers: +, - >
  - Strings – string-length, string=?
- Built-in procedures have names
- The name is evaluated by looking up the procedure in a special table
- Type + into the evaluator window

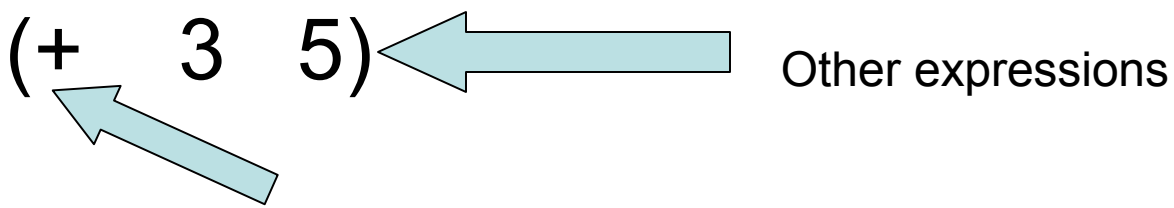
Basic Primitive Procedures

- +, -, =, >, <

- Each procedure generates values of a certain type

Back to 3 + 5

- In 6.001-ese, we compute 3 + 5 by *applying* the primitive addition function to the numbers 3 and 5
- We express this as



Go ahead, type it in!

We can apply these expressions recursively

- (* (+ 3 2) (- 5 1))

Names

- To be able to solve interesting problems, we'll need to be able to do computations with many different types and pieces of data.
- We want to control the complexity of the code we will write.
- We can *abstract* expressions by giving them names

Computing the area of a circle
Which is more readable?

- (* (/ 22 7) 4 4)

```
(define pi (/ 22 7))
(* pi (sq 4))
```

We've used names to represent data and computation. This enables us to abstract away the underlying computation

Evaluating Names

- Think of a table inside the interpreter
- To evaluate a name, the interpreter consults the table

Name	Value
pi	3.14159
+	Procedure
-	Procedure

Adding Names

- To add names to the table, use

  (**define** *name value)*

- **Define** is a *special form*
  - Special form – Something that looks like a combination, but behaves differently

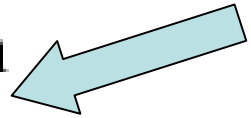
Exercise

- Think of some computation
  - Examples:
    - Area of a rectangle with sides of length 1 and 4
    - Length of hypotenuse of a triangle sides of length 1 and 4
    - Perimeter of a circle of radius 1
- Write out the Scheme Expression to compute it
- Use **define** to make it look cleaner

Scheme Basics

- Rules for evaluation

1. If **self-evaluating,** return value.
2. If a **name,** return value associated with name in environment.
3. If a **special form,** do something special.
4. If a **combination,** then



Substitution Rule

   a. *Evaluate* all of the subexpressions of combination (in any order)

   b. *apply* the operator to the values of the operands (arguments) and return result

(From 6.001 Lecture 1)

Let’s Evaluate an Expression

- (+ 3 5)
- There are four steps that happen

1.
2.
3.
4.

One last example

```
(+ 3 5)                 ➔ 8

(define fred +)         ➔ undef

(fred 4 6)              ➔ 10
```

- How to explain this?

- Explanation
  - + is just a name
  - + is bound to a value which is a procedure
  - line 2 binds the name **`fred`** to that same value

(From 6.001 – Lecture 1)