RACKET

Procedures, Procedure Application,
Procedure Composition and Recursion

Built-in Procedures

- Racket comes preloaded with many procedures.
- Included in them are arithmetic operators
- Have a look at the index of the Dybvig for a list of built-in procedures:

  http://www.scheme.com/tspl4/summary.html#./summary:h0

Procedure Application

- A procedure typically needs arguments.
- A procedure with no arguments is called a *thunk.*
- A procedure is applied to arguments.
- We use a list to apply a procedure.
- Examples:
  - `(+ 3 4)` The procedure + is applied to arguments 3 and 4.
  - `(cons 'a '())` The procedure cons is applied to arguments 'a and '().

Procedure Composition

- In recursive languages such as Racket, we typically use procedure composition to compute results.
- Examples:
  - `(+ 3 (- 7 2))`
  - `(cons (car '(a b)) (cdr '(c d)))`

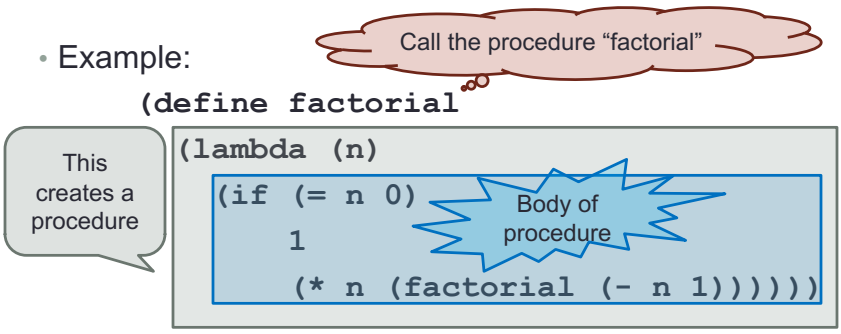
Evaluation of Racket Expressions

- Consider: (+ 3 a)
- Assume a is defined as 7.
- To evaluate the expression we perform the following steps:
  1) Evaluate the 1st item: The procedure +
  2) Evaluate the 2nd item: The number 3
  3) Evaluate the 3rd item: Look up a in our global stash: It is 7
  4) Apply the 1st item (it better be a procedure) to the remaining items:
     Add 3 and 7 and return the result.

Procedure Definition

- To define our own procedures, we use two tools:
  1) A global **`define`**, to add it to the stash of preloaded procedures
  2) A Lambda expression, to make a procedure

- Example:



Class Demo

- Different versions of Factorial:
  - Recursive
  - Tail recursive
- Tracing recursive procedures

Class Exercise

- **(square-sum n)**. Returns the sum of squares of the first n positive integers.
- **(square-all lon)**. Returns a list of the squares of the numbers in lon.
- **(make-list n obj)**. Returns a list of n copies of object.