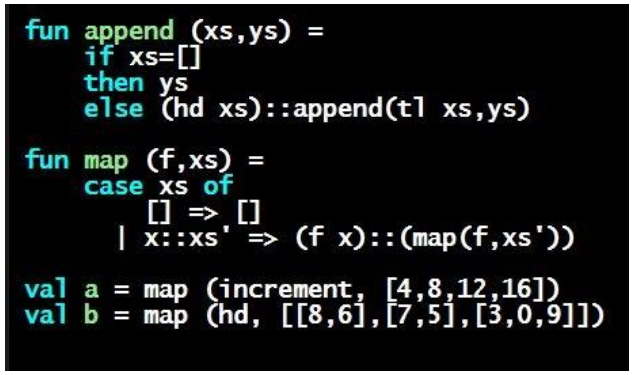

Programming Languages

Dan Grossman

Racket Functions As “Macros”

For Interpreted Language

Recall…

Our approach to language implementation:

- Implementing language *B* in language *A*
- Skipping parsing by writing language *B* programs directly in terms of language *A* constructors
- An interpreter written in *A* recursively evaluates

What we know about macros:

- Extend the syntax of a language
- Use of a macro expands into language syntax before the program is run, i.e., before calling the main interpreter function

Put it together

With our set-up, we can use language *A* (i.e., Racket) *functions* that produce language *B* abstract syntax as language *B* "macros"

- Language *B* programs can use the "macros" as though they are part of language *B*
- No change to the interpreter or struct definitions
- Just a programming idiom enabled by our set-up
  - Helps teach what macros are
- See code for example "macro" definitions and "macro" uses
  - "macro expansion" happens before calling **`eval-exp`**

Optional: Hygiene issues

- Earlier we had (optional) material on hygiene issues with macros
  - (Among other things), problems with shadowing variables when using local variables to avoid evaluating expressions more than once

- The “macro” approach described here does not deal well with this