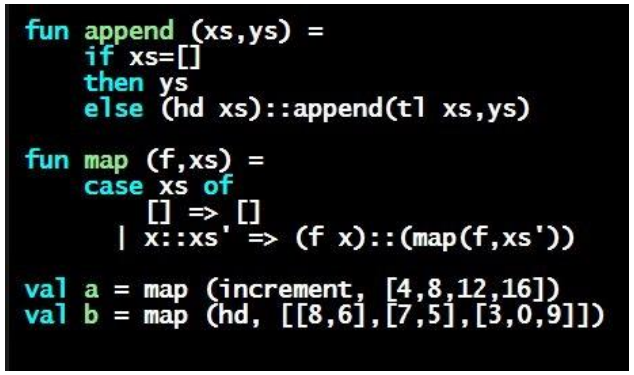

Programming Languages

Dan Grossman
2013

Top-Level Bindings

Top-level

The bindings in a file work like local defines, i.e., **`letrec`**

- Like ML, you can *refer to* earlier bindings
- Unlike ML, you can also *refer to* later bindings
- But refer to later bindings only in function bodies
  - Because bindings are *evaluated* in order
  - Will get an error if you access an uninitialized variable
- Unlike ML, cannot define the same variable twice in module
  - Would make no sense: cannot have both in environment

REPL

Unfortunate detail:

- REPL works slightly differently
  - Not quite **`let*`** or **`letrec`**
  - ☹
- Best to avoid recursive function definitions or forward references in REPL
  - Actually okay unless shadowing something (you may not know about) – then weirdness ensues
  - And calling recursive functions is fine of course

Optional: Actually…

- Racket has a module system with interesting difference from ML
  - Each file is implicitly a module
    - Not really "top-level"
  - A module can shadow bindings from other modules it uses
    - Including Racket standard library
  - So we could redefine **+** or any other function
    - But poor style
    - Only shadows in our module (else messes up rest of standard library)

- (Optional note: Scheme is different)