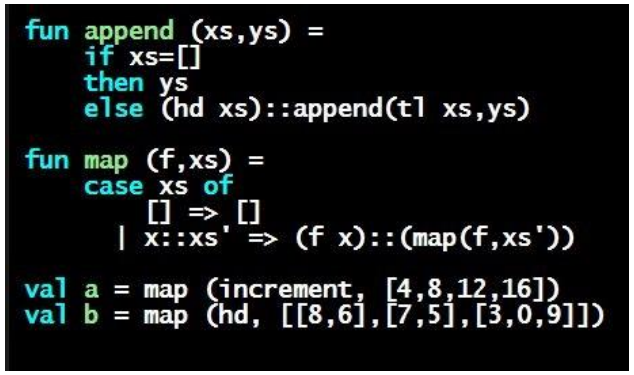

Programming Languages

Dan Grossman

Implementing Closures

The best part

- The most interesting and mind-bending part of the homework is that the language being implemented has first-class closures
  - With lexical scope of course

- Fortunately, what you have to implement is what we have been stressing since we first learned about closures…

Higher-order functions

The “magic”: How do we use the “right environment” for lexical scope when functions may return other functions, store them in data structures, etc.?

Lack of magic: The interpreter uses a closure data structure (with two parts) to keep the environment it will need to use later

```
(struct closure (env fun) #:transparent)
```

Evaluate a function expression:

- A function is *not* a value; a closure *is* a value
  - Evaluating a function returns a closure
- Create a closure out of (a) the function and (b) the current environment when the function was evaluated

Evaluate a function call:

- …

Function calls

```
(call e1 e2)
```

- Use current environment to evaluate **`e1`** to a closure
  - Error if result is a value that is not a closure
- Use current environment to evaluate **`e2`** to a value
- Evaluate closure's function's body in the closure's environment, extended to:
  - Map the function's argument-name to the argument-value
  - And for recursion, map the function's name to the whole closure

This is the same semantics we learned a few weeks ago "coded up"

Given a closure, the code part is *only* ever evaluated using the environment part (extended), *not* the environment at the call-site