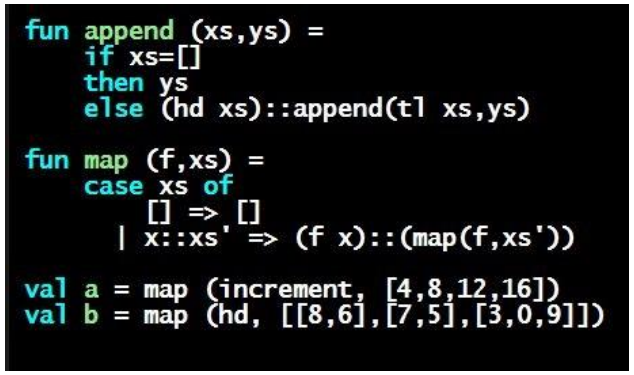

Programming Languages

Dan Grossman

Definition of Lexical Scope

Very important concept

- We know function bodies can use any bindings in scope
- But now that functions can be passed around: In scope where?

Where the function was defined
(not where it was called)

- This semantics is called *lexical scope*
- There are lots of good reasons for this semantics (why)
  - Discussed after explaining what the semantics is (what)
  - Later in course: implementing it (how)
- Must “get this” for homework, exams, and competent programming

Example

Demonstrates lexical scope even without higher-order functions:

```
(* 1 *) val x = 1
(* 2 *) fun f y = x + y
(* 3 *) val x = 2
(* 4 *) val y = 3
(* 5 *) val z = f (x + y)
```

- Line 2 defines a function that, when called, evaluates body **`x+y`** in environment where **`x`** maps to **`1`** and **`y`** maps to the argument
- Call on line 5:
  - Looks up **`f`** to get the function defined on line 2
  - Evaluates **`x+y`** in current environment, producing **`5`**
  - Calls the function with **`5`**, which evaluates the body in the old environment, producing **`6`**

Closures

How can functions be evaluated in old environments that aren't around anymore?

- The language implementation keeps them around as necessary

Can define the semantics of functions as follows:

- A function value has two parts
  - The code (obviously)
  - The environment that was current when the function was defined
- This is a "pair" but unlike ML pairs, you cannot access the pieces
- All you can do is call this "pair"
- This pair is called a *function closure*
- A call evaluates the code part in the environment part (extended with the function argument)

Example

```
(* 1 *) val x = 1
(* 2 *) fun f y = x + y
(* 3 *) val x = 2
(* 4 *) val y = 3
(* 5 *) val z = f (x + y)
```

- Line 2 creates a closure and binds **`f`** to it:
  - Code: "take **`y`** and have body **`x+y`**"
  - Environment: "**`x`** maps to **`1`**"
    - (Plus whatever else is in scope, including **`f`** for recursion)

- Line 5 calls the closure defined in line 2 with **`5`**
  - So body evaluated in environment "**`x`** maps to **`1`**" extended with "**`y`** maps to **`5`**"

Coming up:

Now you know the rule: *lexical scope*.

Next steps (rest of section):

- (Silly) examples to demonstrate how the rule works with higher-order functions
- Why the other natural rule, *dynamic scope*, is a bad idea
- Powerful *idioms* with higher-order functions that use this rule
  - Passing functions to iterators like **`filter`**
  - Several more idioms