Today

- Local Bindings
- Delaying evaluation: Function bodies evaluated only at application
  - Conditionals
  - Streams
  - Laziness
  - Memoization
- In general, evaluation rules defined by language semantics
  - Some languages have “lazy” function application!

Local bindings

There are 3 forms of local bindings with different semantics:

- `let`
- `let*`
- `letrec`

Also, in function bodies, a sequence of definitions is equivalent to `letrec`.

But at top-level redefinition is assignment!

This makes it ghastly hard to encapsulate code, but in practice:

- People assume non-malicious clients
- Implementations provide access to “real primitives”

For your homework, assume top-level definitions are immutable.

Delayed Evaluation

For each language construct, there are rules governing when subexpressions get evaluated. In ML, Scheme, and Java:

- Function arguments are “eager” (call-by-value)
- Conditional branches are not

We could define a language in which function arguments were not evaluated before call, but instead at each use of argument in body. (call-by-name)

- Sometimes faster: `(lambda (x) 3)`
- Sometimes slower: `(lambda (x) (+ x x))`
- Equivalent if function argument has no effects/non-termination
Thunks

One (among several) meanings of “thunk” is just a function taking no arguments, which works great for delaying evaluation.

- Instead of passing a value directly, pass a thunk (function) which yields the value when it is called

If thunks are lightweight enough syntactically, why not make it eager? (Smalltalk does this!)