

predicates

1

Conditionals

- `#t #f`
- *primitives*: `<= >` *not* *and* *or*
- *(if <predicate> <consequent> <alternative>)*
- *(cond (<pred1> <clauses>) (<pred2> <clauses>)...)*

2

- `cons`
- `car`
- `cdr`

3

- `null?`
- `pair?`
- `list?`

4

Procedures as black boxes

- *user only sees what is necessary to see*
- *“what does it do?” not “how”*
- *procedural abstraction*
- *procedural decomposition*
- *sqrt-iter*

5

Block Structure

- *nested definitions*
 - *always come first! (otherwise, behavior is not guaranteed)*
- *hide what doesn't need to be seen*

6

Bound variables

- *procedure def binds formal parameters*
- *consistent renaming*

7

Scope

- *where a name is bound*
 - *bound vars have procedure as their scope*
- *free variables*
- *lexical scoping: when a free var in a procedure is interpreted in the context of an enclosing procedure (used in Scheme, not all Lisps)*
- *sqrt-block2.scm*

8

Recursive Processes

- *fact-rec*
- state is "hidden" in chain of deferred operations
- *fact-rec* is a linear recursive process
(# of frames on stack is $O(n)$)

9

Iterative Processes

- state can be stored in fixed # vars,
so don't need to grow stack
- no deferred operations
- *fact-iter*

10

Recursive Process or

11

Tail Recursion in Scheme

- iterative process with a recursive call
- Procedure is recursive, but process is iterative
 - procedure: what we write
 - process: how machine implements
- Scheme performs in *fact-iter* in constant space, i.e. $O(1)$ space
- Efficient

12