6.001: Get Ready!

IAP 2004, By Ben Vandiver

Glossary

- **program** collection of procedures and static data that accomplishes a specific task.
- **procedure** a piece of code that when called with arguments computes and returns a result; possibly with some side-effects. In scheme, procedures are normal values like numbers.
- **function** see procedure; they're equivalent in scheme. Some other languages make a distinction.
- **argument** An input variable to a procedure. A new version of the variable is created every time the procedure is called.
- **parameter** see argument.
- **expression** A single valid scheme statement. `5`, `(+ 3 4)`, and `(if (lambda (x) x) 5 (+ 3 4))` are expressions.
- **value** The result of a evaluating an expression. 5, 7, and 5 respectively.
- **type** Values are classified into types. Some types: numbers, booleans, strings, lists, and procedures. Generally, types are disjoint (any value falls into exactly one type class).
- **call** Verb, the action of invoking, jumping to, or using a procedure.
- **apply** Calling a procedure. Often used as "apply procedure p to arguments a1 and a2."
- **pass** Usage "pass X to Y." When calling procedure Y, supply X as one of the arguments.
- **side-effect** In relation to an expression or procedure, some change to the system that does not involve the expression's value.
- **iterate** To loop, or "do" the same code multiple times.
- **variable** A name that refers to a exactly one value.
- **binding** Also verb "to bind". The pairing of a name with a value to make a variable.
- **recurse** In a procedure, to call that same procedure again.

Scheme

1. **Basic Elements**

   (a) *self-evaluating* - expressions whose value is the same as the expression.

(b) *names* - Name is looked up in the symbol table to find the value associated with it. Names may be made of any collection of characters that doesn't start with a number.

2. **Combination**
   `(` *procedure arguments-separated-by-spaces* `)`
   Value is determined by evaluating the expression for the procedure and applying the resulting value to the value of the arguments.

3. **Special Forms**

   (a) *if* - `(if` *test consequent alternative*`)`
   If the value of the test is <u>not false (#f)</u>, evaluate the consequent, otherwise evaluate the alternative.

   (b) *define* - `(define` *name value*`)`
   The name is bound to the result of evaluating the the value.

   (c) *lambda* - `(lambda` *parameters body*`)`
   Creates a procedure with the given parameters and body. Parameters is a list of names of variables. Body is one or more scheme expressions. When the procedure is applied, the body expressions are evaluated in order and the value of the last one is returned.

   (d) *let** - `(let*` *bindings body*`)`
   Binds the given bindings for the duration of the body. The bindings is a list of `(`*name value*`)` pairs. The body consists of one or more expressions which are evaluated in order and the value of last is returned.

Programming

1. define the problem
2. describe approach in english ("what's the plan?")
3. write code
4. test it to see if it works

Problems

1. Write a procedure to square numbers
   plan:

2. Write a procedure to sum the squares of two numbers
   plan:

3. Sum the numbers between 1 and some number n
   plan:

4. Compute n!
   plan:

5. Compute e
   plan:

6. Compute nth fibonacci number
   plan:

7. Compute golden ratio
   plan:

8. Compute pi
   plan: