0.1 Some Introductory Lisp

List Constructing Functions

`(cons arg1 arg2)` – constructs a new list whose first element is the value of arg1 and whose remainder is the value of arg2. In essence, it puts the value of arg1 as the first element of the list which is the value of arg2.

List Selector Functions

`(car list)` – returns the first element of the `list` once it has been evaluated.

`(cdr list)` – returns the evaluated `list` with all but its first element.

Giving Values to Variables

We can give values to variables – typically we don't use this function when we are programming, rather we use it a lot when we are testing functions and get tired of writing out long expressions.

`(setf var-name s-exp)` – this function is special in the way that the arguments are evaluated. In particular, the first argument IS NOT EVALUATED, while the second argument IS EVALUATED. The function sets `var-name` to the value of `s-exp`. E.g.,

```
> a
>>Error: The symbol A has no global value.

SYMBOL-VALUE:
   Required arg 0 (S): A
:C  0: Try evaluating A again
:A  1: Abort to Lisp Top Level

-> 1
Abort to Lisp Top Level
Back to Lisp Top Level

> (setf a '(a b c))
(A B C)
> a                    ; note we can now type a and it has a value!
(A B C)
> (setf d '(d e f))
(D E F)
> (cons (car a) (cddr d))
(A F)
```

0.1.1 Defining your own programs

(Wilensky Chapter 3)

All programs in lisp are functions – they are given 0 or more arguments and return 1 value (in general). In addition, some of these functions may have side-effects.

Lisp gives us a special function for defining functions:

```
(defun fn-name (arg-list)
               exp-1
                .
                .            function body
                .
              exp-n)
```

`defun` is a special function whose arguments are not evaluated. The value of the function happens to be the function name, but that is not really important. `defun` has an important side-effect – `fn-name` is defined as a function whose formal parameters are `arg-list` and whose body is `exp-1...exp-n`.

0.1.2 Flow of Control

(Wilensky Chapter 4)

We can't do too many interesting things yet, we would like some functions that give you some control over how things are evaluated. The most important and widely used such function is the `cond` which is similar to a case statement or extended if-then-else statement.

`(cond` $(e_{11}\ e_{12}\ \ldots\ e_{1n_1})$
$(e_{21}\ e_{22}\ \ldots\ e_{2n_2})$
.
.
.
$(e_{m1}\ e_{m2}\ \ldots\ e_{mn_m}))$

where the n's ≥ 1 and m ≥ 1.

`cond` is a special function. It evaluates e_{11}, e_{21}, ... in turn until one is found that is non-nil. Call that e_{k1}. Then the expressions $e_{k2}\ldots\ e_{kn_k}$ are evaluated. The value of e_{kn_k} is returned as the value of the `cond`. Note, if none of the e_{1i}'s evaluate to non-nil, then nil is returned as the value.

*** notice the convention in lisp. false=NIL, true=everything else.