

Mapping

mapcar

Applies a function to each element of a list(s) and returns a list of the results

```
(mapcar #'print '(1 2 x))
```

will print

```
1  
2  
x
```

and return the list

```
(1 2 x)
```

because print returns its argument

```
(mapcar #'list '(1 2 x))
```

will return

```
((1) (2) (x))
```

because list returns a list of its arguments

maplist

Applies a function to each tail of a list(s) and returns a list of the results

```
(maplist #'print '(1 2 x))
```

will print

```
(1 2 x)  
(2 x)  
(x)
```

and return the list

```
((1 2 x) (2 x) (x))
```

because print returns its argument

```
(maplist #'list '(1 2 x))
```

will return

```
((1 2 x) ((2 x)) ((x)))
```

because list returns a list of its arguments

Errors

error

```
error datum &rest arguments => [[no result]]
```

datum is a format string that interprets the arguments to print an error message before entering the debugger

You can print error messages with

```
(error "Bad expression ~a" exp)
```

and the ~a will be replaced with a printout of exp (similar to printf)

Each ~a in the string is replaced with a printout of the next argument

Hints for Lab3

```
(defun diff (exp var)
  (cond ((numberp exp)
        0)
        ((symbolp exp)
         (if (eq exp var)
             1
             0))
        ((consp exp)
         (case (car exp)
             (+ (cons '+
                       (mapcar #'(lambda (x)
                                   (diff x var))
                               (cdr exp))))
             (- ...)
             (* (cons '+
                       (maplist
                        #'(lambda (x)
                            (cons '*
                                   (maplist
                                    #'(lambda (y)
                                        (if (eq x y)
                                            (diff (car y)
                                                var)
                                            (car y)))
                                    (cdr exp))))
                        (cdr exp))))))
             (/ ...)
             (exp ...)
             (log ...)
             (otherwise (error "Bad expression ~a"
                               exp))))))
```