

# CS-257L

## Nonimperative Programming: Scheme!

Instructor:

Joel Castellanos

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering  
Center (FEC) room 321

Scheme and the  
Art of  
Programming  
Chapter 7

# Scheme and the Art of Programming Ch 7

---

- (`map` `proc` `list1` `list2` ...)
- (`for-each` `proc` `list1` `list2` ...)
- (`apply` `proc` `arg1` ... `args`)

# (map proc list1 list2 ...)

- The lists must be lists, and proc must be a procedure taking as many arguments as there are lists and returning a single value. If more than one list is given, then they must all be the same length. Map applies proc element-wise to the elements of the lists and returns a list of the results, in order. The dynamic order in which proc is applied to the elements of the lists is unspecified.

- `(map odd? '(1 2 3 4 5 6 7 8))`  
`(#t #f #t #f #t #f #t #f)`

- `(map + '(1 2 3) '(4 5 6))`  
`(5 7 9)`

## (for-each proc list1 list2 ...)

- The arguments to for-each are like the arguments to map, but for-each calls proc for its side effects rather than for its values. Unlike map, for-each is guaranteed to call proc on the elements of the lists in order from the first element(s) to the last, and the value returned by for-each is unspecified.

- `(for-each odd? '(1 2 3 4 5))`  
`;output undefined`

- `(for-each display '(1 2 3 4 5))`  
`12345`

# WriteIn with for-each

---

```
(define writeIn
  (lambda args
    (for-each display args)
    (newline)
  )
)
```

# (apply proc arg1 ... args)

- Proc must be a procedure and args must be a list. Calls proc with the elements of the list (append (list arg1 ...) args) as the actual arguments.

- (max 24 22 27 21)

27

- (max '(24 22 27 21))

max: expects argument of type <real number>;  
given (24 22 27 21)

- (apply max '(24 22 27 21))

27

# Exercise 7.7 reduce

- Define a procedure, `reduce`, that has two parameters, *proc* and *mylist*.
- The procedure *proc* takes two arguments.
- The procedure `reduce` reduces the list *mylist* by successively applying this operation: it builds a new list with the first two elements of the preceding list replaced by the value obtained when *proc* is applied to them.
- When the list is reduced to containing only two elements, the value returned is the value of *proc* applied to those two elements.
- If the original list contains fewer than two elements, an error is reported.
- Here is how the successive stages in the reduction look for:

`(reduce + '(3 5 7 9)):`

`(3 5 7 9) → (8 7 9) → (15 9) → 24`