1 Learning Goals

- Learn the basics of Scheme
- Begin to see that Scheme is, in fact, beautiful
2 What Would Scheme Display

2.1 What would Scheme display? As a reminder, the built-in quotient function performs floor division.

```
scm> (define a (+ 1 2))
scm> a
```

```
scm> (define b (- (+ (* 3 3) 2) 1))
scm> (= (modulo b a) (quotient 5 3))
```

2.2 What would Scheme display?

```
scm> (cons 10 (cons 11))
scm> (car (cons 10 (cons 11 nil)))
scm> (cdr (cons 10 (cons 11 nil)))
scm> (cons 5 '(6 7 8))
```

```
scm> (define a 10)
 a
scm> (list 8 9 a 11) ; list procedure evaluates all operands
scm> '(8 9 a 11) ; quote special form does not evaluate operand
```

```
scm> (list? (cons 1 2))
scm> (list? (cons 1 (cons 2 '())))
```

```
scm> (define null nil)
scm> (equal? null 'null)
scm> (equal? nil 'null)
scm> (equal? null 'nil)
```

*Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.*
scm> (equal? nil 'nil)

scm> (equal? 'nil ''nil)

scm> (equal? ''nil ''nil)

scm> (eqv? ''nil ''nil)
3 Intro-Level Practice

3.1 Write a function that returns the factorial of a number.

(define (factorial x))

3.2 Define reduce, where the first argument is a function that takes two arguments, the second is a starting value, and the third is a list. This should work like Python’s reduce.

(define (reduce fn s lst) )
4 Exam-Level Prep

4.1 Write a function that takes a procedure and applies to every element in a given nested list.

The result should be a nested list with the same structure as the input list, but with each element replaced by the result of applying the procedure to that element.

Use the built-in list? procedure to detect whether a value is a list.

(define (deep-map fn lst)

scm> (deep-map (lambda (x) (* x x)) '1 2 3)
(1 4 9)
scm> (deep-map (lambda (x) (* x x)) '1 (4) 5) 9)
(1 ((16) 25) 81)

4.2 Fall 2019 Final, Question 7a: Mull It Over Implement mulxy, which multiplies integers x and y. Hint: (- 2) evaluates to -2.

;; multiply x by y (without using the * operator).
;; (mulxy 3 4) -> 12 ; 12 = 3 + 3 + 3 + 3
;; (mulxy (- 3) (- 4)) -> 12 ; 12 = -( -3 + -3 + -3 + -3 )
(define (mulxy x y)

   (cond ((< y 0) (- ________________________________ )

   ((= y 0) 0)

   (else ( ________________ x (mulxy x ________________________ ))))))