Scheme
- Functional programming language
- Based on LISP
- Values: True/False, numbers, lists
- No direct true/false
- Lambda expressions
- First order functions
- Higher order functions
- Resembles lambda calculus
- Based on LISP
- Steele & Sussman @ MIT, 1975

Scheme Examples
- Lists
  - (define s (cons 2 3))
  - (car s) evaluates to 2
  - (cdr s) evaluates to 3
- Higher order functions
  - (define (map f l) (map f l))
- Booleans
  - (or #t #f) evaluates to #t
- (and #t #f) evaluates to #f

Scheme
- Type value
  - Represents Scheme values
  - (define x 2)
  - (car x) evaluates to 2
  - (cdr x) evaluates to #f

Starting OCaml Code – scheme.ml
- Type value
  - Represents Scheme values
  - (define x 2)
  - (car x) evaluates to 2
  - (cdr x) evaluates to #f

Scheme
- Type value
  - Represents Scheme values
  - (define x 2)
  - (car x) evaluates to 2
  - (cdr x) evaluates to #f

Starting OCaml Code – scheme.ml
- Type value
  - Represents Scheme values
  - (define x 2)
  - (car x) evaluates to 2
  - (cdr x) evaluates to #f

Project 4 - Scheme Parser & Interpreter
- Scheme parser
  - Given scanner
  - Converts token strings into sequence of Scheme abstract syntax tree
  - Higher order functions
    - (define fact (lambda (n) (if (= n 0) 1 (* n (fact (- n 1))))))
  - (- 3 4 5) evaluates to -6
  - (- 3 4) evaluates to -1
  - (+ 1 2 3 4 5) evaluates to 15
  - (+ 1 2) evaluates to 3

Project 4 - Part 1
- Scheme programming
  - Gain experience with Scheme programs
  - Write simple recursive functions
    - (define (adder n) (+ n (adder (- n 1))))
    - (adder 5) evaluates to 10
  - Write non-recursive Scheme functions
    - (define (add-two n) (+ n 2))
    - (add-two 5) evaluates to 7
  - (define (double x) (* x 2))
    - (double 5) evaluates to 10
  - (define (two x) (if (= x 0) 0 (* x 2)))
    - (two 5) evaluates to 32

Project 4 - Part 2
- Scheme programming
  - Gain experience with Scheme programs
  - Higher order functions
    - (define (adder x) (+ x (adder (- x 1))))
    - (adder 5) evaluates to 10

Project 4 - Part 3
- Scheme parser
  - Given scanner
  - Converts token strings into sequence of Scheme abstract syntax tree
  - Write non-recursive Scheme functions
    - (define (add-two n) (+ n 2))
    - (add-two 5) evaluates to 7

Project 4 Notes
- Project Notes
  - Basic setup — your code for part 1
  - Scheme ml — your code. Make all your edits here
  - Park ml — interpreter using code from scheme.ml
  - Sample output — sample input / output (public test)
  - Testing
    - Run scheme.ml - test for syntax
    - Test scheme.ml - test scheme interpreter
    - Test public.wml — test 3rd party public test
    - # OCaml