Homework 2 Due February 18th 11:59 p.m.

You should submit a single .scm file with all definitions on learn

All answers which are not Scheme definitions should be written using comments (;)

- Define a function *deepen-n* that takes two parameters, *ls* and *n*. This function should wrap *n* pairs of parens around each top level element in *ls*. For example:
  - (deepen-n '() 10) => ()
  - (deepen-n '(1 2 3 4 5) 0) => (1 2 3 4 5)
  - (deepen-n '(1 2 3 4 5) 1) => ((1) (2) (3) (4) (5))
  - (deepen-n '(1 2 3 4 5) 2) => (((1)) ((2)) ((3)) ((4)) ((5)))
  - (deepen-n '(1 2 3 4 5) 3) => ((((1))) (((2))) (((3))) (((4))) (((5))))
  - (deepen-n '((1 2) (3 4) ((5) 6)) 2) => ((((1 2))) (((3 4))) ((((5) 6))))
- Define a function *insert-left-all* that takes three parameters, *new old ls*. This function should insert *new* to the left of every occurrence of *old* in *ls*. This function should recurse into sublists to find *all* occurrences of *old*. For example:
  - (insert-left-all 'z 'a'()) => ()
  - (insert-left-all 'z 'a'(a ((b a) ((a (c)))))) => (z a ((b z a) ((z a (c)))))
  - (insert-left-all 'z 'a'(((a)))) => (((z a)))
- Define two functions *mk-asc-list-of-int* and *mk-desc-list-of-ints* iteratively (tail recursion). These function should take a *single* argument *n*. They should produce either an ascending list from 1 to *n* or a descending list from *n* to 1. Hint: Use a helper function. For example:
  - (mk-asc-list-of-ints 0) => ()
  - (mk-asc-list-of-ints 1) => (1)
  - (mk-asc-list-of-ints 5) => (1 2 3 4 5)
  - (mk-desc-list-of-ints 0) => ()
  - (mk-desc-list-of-ints 1) => (1)
  - (mk-desc-list-of-ints 5) => (5 4 3 2 1)
- Define a function *calculator* that takes one argument *expr*. This function should evaluate the infix *expr* and return its value. You can assume that all sub-expressions are parenthesized (no need to worry about precedence), will contain only natural numbers, and will only contain the four basic math operators, *+*, *-*, ***, */*. For example:
  - (calculator 42) => 42
  - (calculator '(1 + 2)) => 3
  - (calculator '(1 + (2 * 8))) => 17
  - (calculator '((((2 + 3) * 2) / 5) + (17 - 1))) => 18
- Define a function *infix->prefix* that takes on argument *expr*. This function must return a prefix expression corresponding to the infix *expr* argument. For example:
  - (infix->prefix 42) => 42
  - (infix->prefix '(1 + 2)) => (+ 1 2)

  - (infix->prefix '(1 + (2 * 8))) => (+ 1 (* 2 8))
  - (infix->prefix '((((2 + 3) * 2) / 5) + (17 - 1))) => (+ (/ (* (+ 2 3) 2) 5) (- 17 1))
- Define a function iota-iota that takes an argument *n*. This function must return a list of **pairs** of integers such that:
  - (iota-iota 1) => ((1 . 1))
  - (iota-iota 2) => ((1 . 1) (1 . 2) (2 . 1) (2 . 2))
  - (iota-iota 3) => ((1 . 1) (1 . 2) (1 . 3) (2 . 1) (2 . 2) (2 . 3) (3 . 1) (3 . 2) (3 . 3))
  - Any helper functions should be tail-recursive and defined within the body of *iota-iota* using a *letrec*.
  - Hint: Define a function *iota* which takes an argument *n* and returns a list in the following range: [1, *n*]. Remember this function must ultimately be defined within *iota-iota*.
- Define a **tail-recursive** function *digits->number* that takes a list of digits, *ds*, and returns the number represented by those digits. For example:
  - (digits->number '(1 2 3 4)) => 1234
  - (digits->number '(7 6 1 5)) => 7615
  - Any helper functions should be tail-recursive and defined within the body of *digits->number* using a *letrec*.
- Define a function *cond->if* that takes a *cond* expression, *expr*, as an argument and transforms it into an equivalent *if* expression. For example:
  - (cond->if '(cond ((> x y) (- x y)) ((< x y) (- y x)) (else 0))) => (if (> x y) (- x y) (if (< x y) (- y x) 0))
- Define a function *sine* (note the *e* at the end) which takes a number x as an argument and returns *sin(x)*. This function must approximate *sin* using the first 100 terms of the Taylor series for *sin*. This series is given as follows: $sin(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots$
  - Any helper functions should be defined within the body of *sine* using *letrec*.
  - Extra Credit: Define this function without using or reinventing *expt* or *fact*.