CSSE 304 Day 01 Summary and class notes (like the in-class quizzes in other courses, but you don't turn them in)

Announcements are linked from the schedule page (Resources column). A small assignment is due TODAY at 11:59 PM!

Important links and addresses for the course:

Schedule page: http://www.rose-hulman.edu/class/csse/csse304/
Grading server: https://plc.csse.rose-hulman.edu/
Piazza: Announcements, Questions and answers, bug reports: https://piazza.com/rose-hulman/fall2019/csse304/
Email: csse304-staff@rose-hulman.edu , anderson@rose-hulman.edu Please include 304 in subject line of any email you send to me.

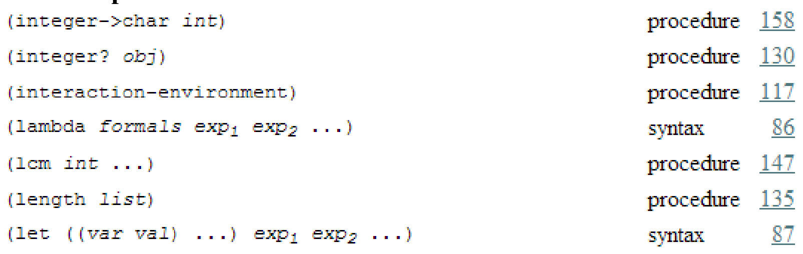
Main ideas from today:

1. Course intro and instructor intro will happen Day 3, so we can dive into Scheme today.
2. About Scheme(lots of stuff printed here, so you don't have to write it down.
   a. For us: Petite *Chez* Scheme from http://scheme.com/ (v 8.4) and https://cisco.github.io/ChezScheme/ (v 9.5).
   b. Scheme was invented in 1975 by Guy Steele and Gerald Sussman.
   c. Syntax is like LISP, semantics more like Algol family (Pascal, Ada, C, Java, C#).
   d. Expression oriented and interactive (like Maple, Python, Matlab). Can also write and compile programs.
   e. Programs and data have the same syntax (lots of parentheses).
   f. Not statically typed (a variable's type is determined by the data it holds, not by a declaration).
   g. Linked lists are the most important data type. Built-in to the language syntax, not an add-on.
   h. Garbage-collected, with no explicit pointers.
   i. Prefix notation for procedure calls: `(+ a (* b c))` instead of `a + b * c`.
   j. Like Java: primitives are passed by value. Others (e.g. lists, vectors): references are passed by value.
   k. Like Maple: symbols are a data type.
   l. Procedures are first-class data (we will explore this in detail soon)
      i. Can pass them as arguments to other procedures.
      ii. A procedure can create and return another procedure.
      iii. Can store a procedure in a data structure (e.g., list or vector of procedures)
   m. Java's ***new*** operator has no Scheme equivalent.
   n. Instead there are specific procedures for creating objects of each type:
      i. **cons** creates a pair
      ii. **vector** creates a vector (i.e. an array)
      iii. **list** creates a list
      iv. **string** creates a string
      v. **lambda** (which is not itself a procedure) creates a procedure
      vi. **define-syntax** (also not a procedure) creates new syntax that extends the Scheme language itself.

Scheme elements from demo: quote, **define**, prefix procedure calls, procedure assigned to a variable, list, cons, '(), null?, equal?, eq?, list-ref, list→vector, vector-ref, car, cdr, cadr, cddr, etc., **let**, **lambda**, **trace**.

Items in **bold-face** are not procedures.

3. The **Summary of Forms** in TSPL can be very useful: http://www.scheme.com/tspl4/summary.html#./summary:h0
   a. In that document, "syntax" basically means "any construct that is not a procedure".
   Examples:



(integer->char *int*)	procedure	158
(integer? *obj*)	procedure	130
(interaction-environment)	procedure	117
(lambda *formals* exp_1 exp_2 ...)	syntax	86
(lcm *int* ...)	procedure	147
(length *list*)	procedure	135
(let ((*var* *val*) ...) exp_1 exp_2 ...)	syntax	87

4. Check your knowledge from the introductory videos. What will Scheme print if we execute each of these expressions?

```
(/ 3 5)
(cadr  ' (a b c))
(cons 1 ' (a b c))
(cons 1 2)
(eq? ' (a b c) '(a b c))
(let ([a 5][b 6])
   (+ 3 a b))
```

5. The rest of class time will be an interactive Scheme session. Feel free to use the rest of this page (including the back) for notes on today's class material, especially the interactive session. Better yet, follow along on your computer. I will post a transcript of what we do today, so don't worry if you can't capture everything. To find that transcript, follow the Live in Class link from the resources column of day 1 of the schedule page.