Lecture 2: Compilers, Interpreters, and Racket.
Agenda:

- what is a programming language?
- how do we execute a program?
- compilers vs. interpreters
- syntax + semantics
- Racket overview
What is a programming language?

way to specify algorithms?

High vs. low level?

unambiguous way of specifying logic

syntax + semantics

grammar → meaning
How do we "execute" a program?

(i.e., how do we go from HLL source code to fetch-decode-execute on the CPU?)

compiler

intermediate

scanning/lexing
Text (source code) -> parsing

Data structure (parse tree/AST) -> static analysis

Intermediate representation -> optimization

Intermediate representation -> code generation

Native/virtual machine code

VM/CPUs/OS
(from "Crafting Interpreters")
"Compiler" vs. "Interpreter"

HLL → some other static form (e.g., bytecode, native code, other lang)

HLL → execution!

lots of middle ground!
(from "Crafting Interpreters")
Syntax + Semantics

- rules for writing a valid (legal) program
- GRAMMAR

- what does a program (or some language construct) mean?
- how will a program behave (what will it compute) when run?
  - result value/type/eventuality
- operational semantics: formal, logical reasoning about a program
  - mathematical foundations
Racket

(a descendant of Scheme, and a member of the USP family of languages)

"LIST Processing"
"Lots of Insidious, Silly Parentheses"
"Lost In a Sea of Parentheses"
#lang racket
(define (quicksort < l)
  (match l
   ['() '()]
   [(cons x xs)
    (let-values ([(xs-gte xs-lt) (partition (curry < x) xs)])
      (append (quicksort < xs-lt)
               (list x)
               (quicksort < xs-gte)))]))}
public static <E extends Comparable<? super E>> List<E> quickSort(List<E> arr) {
    if (arr.isEmpty())
        return arr;
    else {
        E pivot = arr.get(0);

        List<E> less = new LinkedList<E>();
        List<E> pivotList = new LinkedList<E>();
        List<E> more = new LinkedList<E>();

        // Partition
        for (E i: arr) {
            if (i.compareTo(pivot) < 0)
                less.add(i);
            else if (i.compareTo(pivot) > 0)
                more.add(i);
            else
                pivotList.add(i);
        }

        // Recursively sort sublists
        less = quickSort(less);
        more = quickSort(more);

        // Concatenate results
        less.addAll(pivotList);
        less.addAll(more);
        return less;
    }
}

(it's too easy to pick on Java)
"S-expressions"

- atoms: numbers, strings, or "symbols" (identifiers)
- parenthesized list of space-delimited S-expressions

E.g.: 1, 2, "hello world", apple, really?, a-b/c, ( ), (2 3 4), (a b 1), ("hello"), (+ 1 2), (foo 1 2 (bar 3))

# special characters: ( ) [ ] { } " ' , ; # \
(foo 1 2 (bar 4 (bar (bat 8 ))))

(define (foo x y)
  (+ x (* 2 y))))