1. Computability

(a) be able to identify partial functions and what recursive functions
(b) be able to explain what it means for a language to be Turing-complete
(c) be able explain what the Church-Turing Thesis tells us about the computational power of lambda-calculus, Turing Machines, partial \( \mu \)-recursive functions, and different (Turing-complete) programming languages
(d) given a description of a programming languages with recursion or general looping, be able to identify it as being Turing-complete
(e) given a description of a language that always terminates in a finite amount of time, be able to identify it as not being Turing-complete
(f) be able to list several languages that are and are not Turing complete
(g) be able to explain why the Halting Problem is undecidable

2. Lisp/Scheme

(a) be able to read Lisp expressions and read/write Scheme expressions
   i. using the `define`, `lambda`, `cond`, `quote`, `let`, `if`, and `eval` special forms
   ii. using numerical (e.g., `<` and equality comparison (e.g., `eqv?`) operators
   iii. using mathematical functions (e.g., `+`, `sqrt`)
   iv. using list functions (e.g., `cons`, `car`, `cdr`, `list`)
(b) be able to explain some of the important contributions of Lisp to the history of Programming Languages
(c) be able to read/write recursive functions (esp., those that recurse over lists) in Scheme
(d) be able to read/write and use higher-order functions in Scheme
(e) be able to read/write curried functions in Scheme

3. \( \lambda \)-Calculus

(a) be able to explain what \( \lambda \)-calculus is and its theoretical significance
(b) understand \( \lambda \)-calculus terms
   i. know what the three types of \( \lambda \)-calculus terms are and their closest analogues in conventional programming languages (i.e., abstraction \( \sim \) function definition, application \( \sim \) function call)
   ii. be able to correctly partition \( \lambda \)-calculus terms into their constituent subterms (e.g., by creating or identifying the “correct” parse tree according to precedence and associativity)
   iii. be able to identify free variables in \( \lambda \)-calculus terms
   iv. be able to determine whether two \( \lambda \)-calculus terms are \( \alpha \)-equivalent
(c) be able to β-reduce simple λ-calculus terms
(d) be able to recognize normal forms and derive normal forms for simple λ-calculus terms
(e) be able to explain how numbers and boolean conditions can be encoded in curried λ-calculus functions
(f) be able to explain Ω: its concrete λ-term, how it reduces itself forever, and that it is only one of several such λ-terms that do not have a normal form
(g) be able to explain Y: its concrete λ-term, its properties, and how it can be used to build recursive functions

4. Formal Languages, BNF, and Parse Trees

(a) be able to explain what a formal language is and what it is concerned with
(b) be able to list the classes of the Chomsky hierarchy ordered by their level of generality
(c) be able to recognize/write derivations showing some wff is in a language described by a BNF grammar
(d) be able to draw a parse-tree for wff of a language described by a BNF grammar
(e) ambiguity
   i. be able to recognize simple ambiguous grammars when given a concrete wff that has two different parse trees
   ii. be able to apply precedence rules to disambiguate concrete parse trees
   iii. be able to apply associativity rules to disambiguate concrete parse trees

5. Compilers, Interpreters, and Virtual Machines

(a) compiler vs. interpreter
   i. be able to explain what a compiler is
   ii. be able to explain what an interpreter is
   iii. be able to articulate the advantages and disadvantages of compilation relative to
   iv. understand how modern programming language implementations, such as Sun’s implementation of Java, often combine aspects of compilation and interpretation into a single execution environment
   v. be able to explain what a JIT compiler is
(b) be able to explain what happens in each of the compiler phases listed in Mitchell
(c) be able to list the 5 optimizations described in Mitchell