Chapter 5: Functional Programming

- What is the distinction between a formal parameter and actual parameters? (Equivalent wording: between bound variables and arguments)
- Why is adding to a list with cons more efficient than append?
- What is the difference-list technique?
- We won't get into the "shortening" call methods of Scheme shown in table 5.1, because Racket has commands like second, third, fourth, etc. But make sure you understand them, and the take, list-ref, and list-tail functions.
- Use of let, let*, and let-rec functions
- Principles of recursion:
  - Solve for the base case. Solve for *n* in terms of solution for *n*-1.
  - For lists, the base case is an empty list and the recursive step is handled in the else clause. For numeric values, the base case is usually 0 or 1.
  - Use cons rather than append to build lists.
  - Use let to name recomputed subexpressions, to avoid re-evaluation
  - Nest local functions (information hiding)
  - Factor out constant parameters
  - Difference lists technique (send "solution so far" down the recursive chain)
  - Correctness first, simplification second
- Wlaschin's principles of functional programming:
  - Composition everywhere
  - Strive for totality
  - Don't repeat yourself
  - Parameterize the things
  - The Hollywood Principle (don't call us, we'll call you)
- Terms: first-class entity, side effects, lambda expression, S-expression

Chapter 6: Bindings and Scopes

- Your text states "a closure can be though of as a pair of pointers." Where do they point?
- Consider the languages: C++, Java, Racket, Python. For each, does the language use static or dynamic scoping?
- What is an environment? Why is it important?
- Be able to find the lexical address of an expression in a nested function.
- Explain the difference between a bound variable and a free variable. If an expression has no free variables, what does that imply about its semantics?
- Advantages/disadvantages of static scoping; advantages/disadvantages of dynamic scoping. Which have most languages adopted? Why?
- What is the FUNARG problem? Why does it arise in languages where functions are first-class entities?
- How is a closure similar to an object? How is it different? (Note: Slides will probably not be adequate here… you might have to actually read the book….)
- What is the difference between deep binding, shallow binding, and ad hoc binding?
- Terms: closures, first-class closures, static v. dynamic scoping, lexical addressing, upward & downward FUNARG problem, denotation, binding, scope, lexical scoping, scope hole, ancestor blocks.

Chapter 7: Type Systems

- You will not need to read/follow/write Haskell or ML code. You may need to read Racket / Scheme code.
- Distinguish between: type coercion v. type casting; parametric polymorphism; type inference; function overloading v. overriding.
- Advantages of static typing over dynamic typing, and of dynamic over static.
- Terms: static typing, dynamic typing, parametric polymorphism, type inference, function overloading, function overriding, strongly typed, weakly typed, sound v. unsound (safe v. unsafe) type systems, explicit v. implicit typing, explicit v. implicit conversion, monomorphic, type inference.
- Most of the material from this chapter should (I hope) be at least somewhat familiar already, from your earlier programming classes.

Chapter 8. Currying & Higher-order functions

- What is currying? Why is it useful?
- How do higher-order functions expand possibilities in program development?
- Understand the workings of the foldl and foldr functions.(Your text examples are in Haskell, but both functions are present in Scheme and Racket.)
- Slides walk through construction of the implode, string2int, powerset functions. You won't be responsible for the ML syntax, but do look at how the process is done, how higher order functions are used to customize what the generic functions (map, fold, etc) are doing.
- Pay attention to section 8.5, analysis; this summarizes the main points of the chapter without getting into the details of code.
- Terms: Partial application, currying, higher-order functions, first-class closures

Chapter 9: Data Abstraction

- What is the distinction between a discriminated and undiscriminated union?
- What is a variant record? How is it usually implemented?
- Explain the difference between interface & implementation. Which does an application need to be aware of,and which can an application ignore?
- Terms: aggregate data types, inductive data types, abstract syntax, type system, abstract syntax, AST