Final Exam Comments

- **Comprehensive** — over everything we have covered during the semester – nothing below modifies that
- Exam not yet written
- No release of previous final exams or return of graded papers
- Concatenation of the 2018 (and 2017) exams gives some idea of what I consider important and how I might ask questions

  Think about other questions that might be asked in the same areas as previous exams
- I tend to focus on important concepts (including terminology) and *use* of knowledge for problem solving, understanding programs, programming, language design, etc.
- Final likely contains new questions, examples, coding problems, etc.

  Study of previous examples, homeworks, etc. cover the skills
- Programming assignments reinforce ideas from textbook

  Assignment #5 works with case study from chapters 41-44
- Chapter 1: Trends over past 70 years and their implications for today's and tomorrow's languages — key firsts in language design and implementation (Fortran, Lisp, Simula, Smalltalk, ML, Haskell, etc.)
- Chapter 2: Key concepts and terminology from primary programming paradigms (imperative, declarative, functional, logic, modular, procedural, etc.)
- Chapter 3: NOT COVERED (but has key concepts/terminology from object-based and object-oriented programming)

  (Used chapters 2-3 in CSci 556 along with Python 3)
- Chapter 4: Basic Haskell programming notation, concepts, and skills
- Chapter 5: Haskell type system (continued later with lists, polymorphism, first-class functions, abstract data types, type classes, etc.)
- Chapters 6-7: Abstraction — procedural abstraction, stepwise refinement, data abstraction (abstract data types), contracts, interfaces, information hiding, etc.

  Abstraction ideas continued in later chapters, including the ELI Calculator language case study

- Chapter 8: Not covered directly, but covered evaluation ideas informally along with chapter 9 and following (time and space complexity and termination)
- Chapter 9: Recursion styles — backward/forward, linear/nonlinear, tail recursion, accumulating parameters, etc.
- Chapter 10: DOES NOT EXIST YET (input/output)

  Suggests Haskell Wikibook chapter
- Chapters 11-12: NOT COVERED — adds some formality to our informal software testing approach

  (Used chapters 11-12 in 556 along with Pytest)
- Chapters 13-14: First-order polymorphic list programming (including pattern matching)
- Chapters 15-17: Higher-order polymorphic list programming and function concepts
- Chapter 18: List comprehensions
- Chapters 19-20: NOT COVERED

  Draft chapters have further information on problem solving and programming in Haskell
- Chapter 21: Algebraic Data Types and Haskell programming with them
- Chapter 22: Type classes and overloading

  Also covered the *Movable Objects example* with this (see lecture notes)
- Chapter 23: NOT COVERED

  But gives another case study that uses data abstraction (like chapter 7) and algebraic data types and type classes (like chapters 21-22)
- Chapters 24-39: NOT COVERED (several chapters do not exist)
- Chapter 40: Language processing pipeline – Really slides and excerpt from Scott's textbook and from Mitchell's textbook
- Chapters 41-44: Language processing case study of ELI Calculator language interpreter — including ideas of concrete and abstract syntax, informal semantics and evaluation, lexical analysis, parsing, etc.
- Chapters 45-46: NOT COVERED, but continues ELI Calculator language case study ideas
- Chapter 47: DOES NOT EXIST – Next case study on ELI Imperative Core language (with variable and function definition, assignments, loops, function calls, etc.)

- Chapters 48-79: DO NOT EXIST – future chapters on other interpreters
- Chapter 80: NOT COVERED – has some math review