CS61A Lecture #38: Conclusion

Announcements:

- "Homework" 11 will be judging the contest. Due next Friday.
- Contest submissions due next Tuesday at midnight. Submit your `contest.scm` file and Scheme project files (updated if needed) as `proj4contest`.
- HKN surveys TODAY: 5 bonus points for filling out their survey. Be sure to sign the sign-up sheet.
- Please fill out our own final survey by Friday, 16 May (worth 1.5 points). If at least 90% of the class fills it out, *everyone* gets another point! See class web page.
- Andrew is looking for lab assistants for summer 61A. Lab assist to help students (as you were helped), reinforce what you've learned, and become a better teacher. You may lab assist for units, and if you're interested in reading or TAing in the future, this is the first step. See Piazza for details.
- Guerrilla section this Monday at 7pm on Scheme, Logic Programming, and Streams in 271 Soda. Check Piazza for details.

A Summary of Topics

- Programming primitives
- Derived programming structures
- Programming-language concepts, design, and implementation
- Programming “Paradigms”
- Software engineering
- Analysis
- Side excursions
- What’s Next?

Programming Primitives

- Recursion: the all-encompassing repetitive construct; recursive thinking
- Pairs: A universal data-structuring tool.
- Functions as data values, functions on functions
- Exceptions: Dealing with errors.
- Classes.

Derived Programming Structures

- Can build almost anything from primitives.
- Although Python also has specialized implementations of some important data structures.
- Sequences:
  - Lists: traversals, searching, inserting, deleting (destructive and non-destructive)
  - Trees: traversals, binary search trees, constructing, inserting, deleting
- Maps.
- Sequences: creating, traversing, searching,
- Iterators, generators.
- Trees: uses, traversing, and searching.

Programming-Language Concepts, Design, Implementation

- Python was developed largely as a teaching language, and is simpler in many ways than other "production" languages...
- And yet, it is a good deal more powerful (as measured by work done per line of code) than these same languages.
- Still, as you've seen, there are problems, too: dynamic vs. static discovery of errors.
- Big item: scope (what instance of what definition applies to evaluation of an identifier). This is what environment diagrams are intended to model.
  - Alternative: dynamic scoping.
- Implementing a language [CS164]:
  - Interpreters
  - Trees as an intermediate language
  - Relationship of run-time environment representation to scope rules.
  - "Little" languages as a programming tool

Paradigms

- Functional programming: expressions, not statements; no side-effects; use of higher-order functions.
- Streams
- Data-directed and object-oriented programming
  - Organize program around types of data, not functions
  - Inheritance
  - Interface vs. implementation
- Rule-based programming (Prolog)
  - Declarative rather than imperative
  - Rule $\rightarrow$ action idea.
  - Logic programming:
    - Pattern matching, pattern variables as a programming tool
    - Declarative *and* imperative interpretation
    - Application to parsing

Software Engineering

- Biggest ideas: Abstraction, separation of concerns
- Specification of a program vs. its implementation
  - Syntactic spec (header) vs. semantic spec (comment).
  - Example of multiple implementations for the same abstract behavior
- Testing: for every program, there is a test.
  - In "Extreme Programming" there is a test for every module.
- Software engineering implicit in all our software courses, explicit in CS169.

Analysis

- What we can measure when we measure speed:
  - Raw time.
  - Counts of selected representative operations.
  - Symbolic expressions of running time.
  - Best/worst case.
- Application of *asymptotic notation* ($\Theta(\cdot)$, etc.) to summarizing symbolic time measurements concisely.

Important Side Excursions

- User Interfaces: there are principles behind making computers usable.
- Cryptography:
  - protecting integrity, privacy, and authenticity of data.
  - Symmetric (DES, Enigma) and asymmetric (public-key) methods.
- Computatbility [CS172]: Some functions cannot be computed. Problems that are "near" such functions cannot be computed quickly.

What's Next (Course-Wise)?

- CS61B: (conventional) data structures and languages
- CS61C: computing hardware as programmers see it.
- CS170, CS172, CS174: "Theory"—analysis and construction of algorithms, theoretical models of computation, use of probabilistic algorithms and analysis.
- CS161: Security
- CS162: Operating systems.
- CS164: Implementation of programming languages.
- CS160, CS169: User interfaces, software engineering.
- CS188: Artificial intelligence.
- CS184: Graphics.
- CS186: Databases.

What's Next (Course-Wise) (II)

- CS189: Machine Learning.
- CS191: Quantum Computing.
- EE C125: Robotics
- EECS C149: Embedded Systems.
- CS 150: Digital Systems Design
- CS 176: Computational Biology
- CS194: Special topics. (E.g.) parallel software; computer animation; data science; networks, crowds, and markets; cell phones as a computing platform.
- Plus graduate courses on these subjects and more.
- And please don't forget CS199 and research projects.

What's Next (Otherwise)?

- Programming contests.
- The open-source world: Go out and build something!
- And above all: Have Fun!