Conclusion and Ask Me Anything

Announcements

61A Wrapped

Programming Paradigms

We started with **imperative** programming which used statements to change the environment and dictate the control flow of a program.

We dabbled with **functional** programming where we explored functions as tools for building abstraction barriers and abstract data types (like trees).

We moved on to **object-oriented** programming which allowed us to formalize bundles of related data and functions together into classes as attributes and methods.

We returned to **functional** programming more rigorously in Scheme, where the basic building block of the program became functions and expressions.

We ended with **declarative** programming in SQL where we describe the output we want rather the specifying the process to compute it.

Concepts

Environments

Control

Iteration

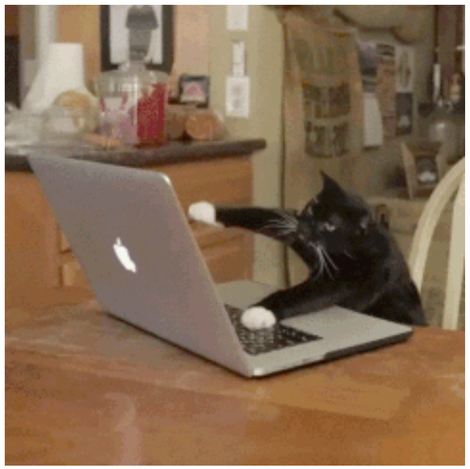
Higher-order Functions

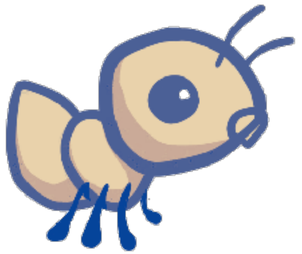
Recursion

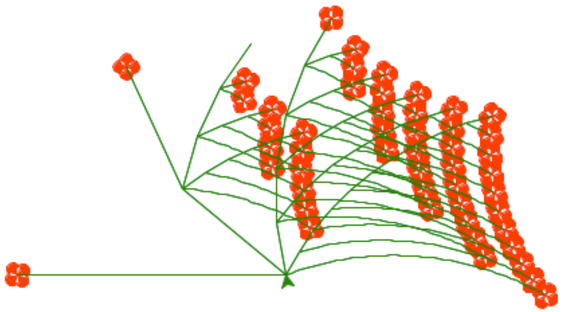
Data Structures: Trees, Linked Lists, Dictionaries

Mutability

Efficiency







What comes next?

CS 61B: *Data Structures & Algorithms* - Java

1. When are Linked Lists more efficient than Arrays? How does lst.sort() work in Python?
2. How is a Dictionary implemented?
3. Projects: Write a chess AI, Build a text editor, Implement Git

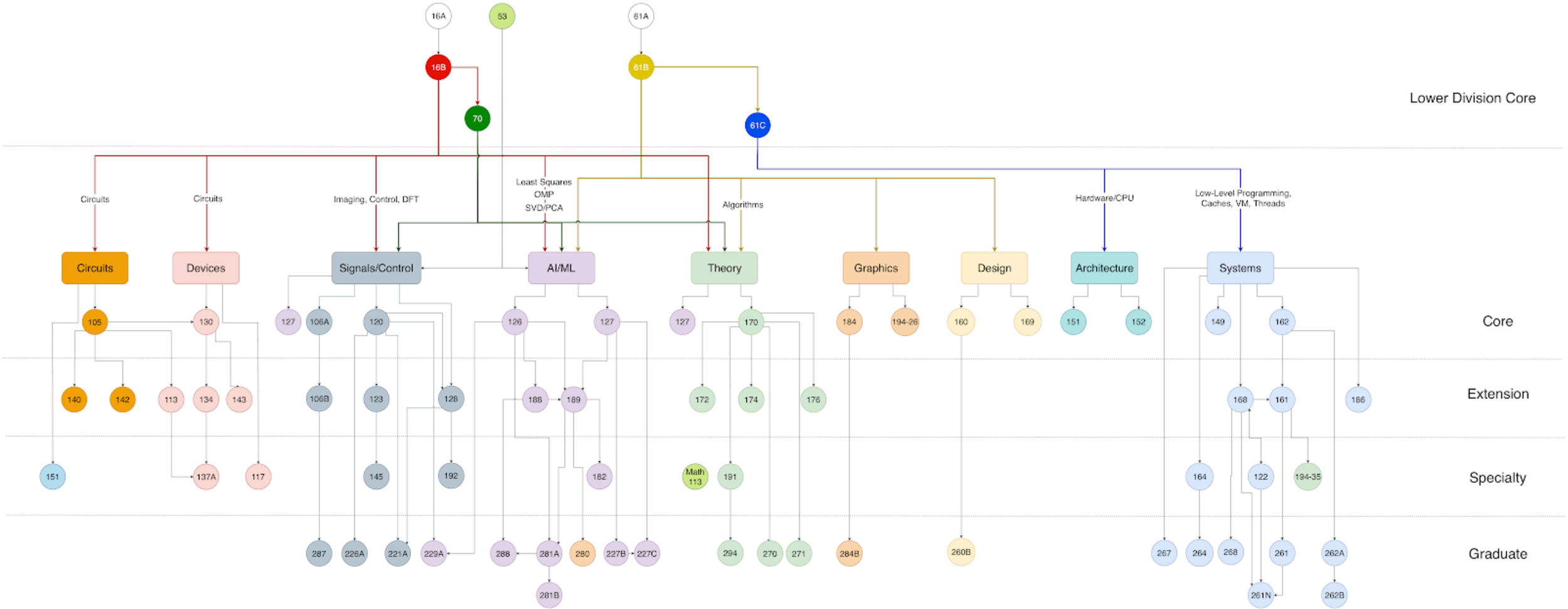
CS 61C: *Computer Architecture* - C, Assembly

1. How does your computer execute your code? What's really happening when you run `python3 hw01.py`?
2. Why is Integer.MAX_VALUE := 2147483647 in Java?
3. Projects: Design your own CPU, Implement NumPy, Program a classification ML algorithm in Assembly Code

CS 70: *Discrete Math & Probability* - Math

1. Can we be certain that our function is correct for all inputs if we only have a finite # of doctests? Yes! You can write a formal mathematical proof.
2. Applications: How do cryptographic encryption algorithms guarantee security?

EECS course map



https://hkn.eecs.berkeley.edu/courseguides

Outside of EECS

https://eecs.berkeley.edu/academics/courses/tech-electives/

Program outside of class

On campus

1. Student organizations. Tons for software engineering, server infrastructure, machine learning, virtual reality, video game development, user interface design, etc.

2. Research. Opportunities within and outside of EECS.

3. Hackathons. Travel and code with friends!

4. Join course staff! EECS TA responsibilities often require coding.

Online

1. Open-source. Python programming language, Linux, Git, ReactJS.

2. Leetcode. Apply problem-solving techniques to new problems.

3. Personal projects. Personal website, messaging bot, web scraper.

Course Staff

TAs!

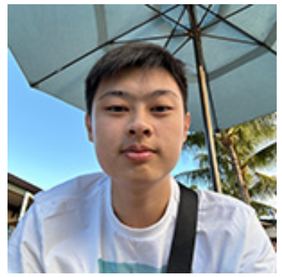


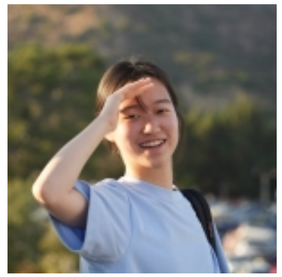





Tutors!







John!

Why join course staff?

Deepen your conceptual understanding, practice debugging, improve technical communication.

Connect with and learn from bright and motivated faculty, other course staff, and *students*.

Give back to Berkeley EECS. Support those that come after you.

Autonomy to explore. TAs have the freedom to experiment in their classroom with different teaching styles, tools, etc.

  – Beyond teaching, TAs in EECS have the opportunity to write problems, develop course software, and run student support programs.

How to join course staff?

Apply to be an Academic Intern:

- Answer questions as a Lab assistant or OH assistant, mentored by a TA.

- Application: Posted on Ed near the start of the next semester.

Apply to be a Tutor:

- Teach tutoring sections, run office hours, answer Ed questions, contribute to course content or software.

- Application: https://deptapps.eecs.berkeley.edu/ase/eecs/apply/

(*Alternative*) **Apply to Computer Science Mentors:**

- Student organization that provides weekly small-group tutoring in the fall and spring.

- Application: https://csmentors.berkeley.edu/

Ask me anything

That's all. Thanks!