CS110 Lecture 1: Introduction

CS110: Principles of Computer Systems

Winter 2021-2022

Stanford University

Instructors: Nick Troccoli and Jerry Cain

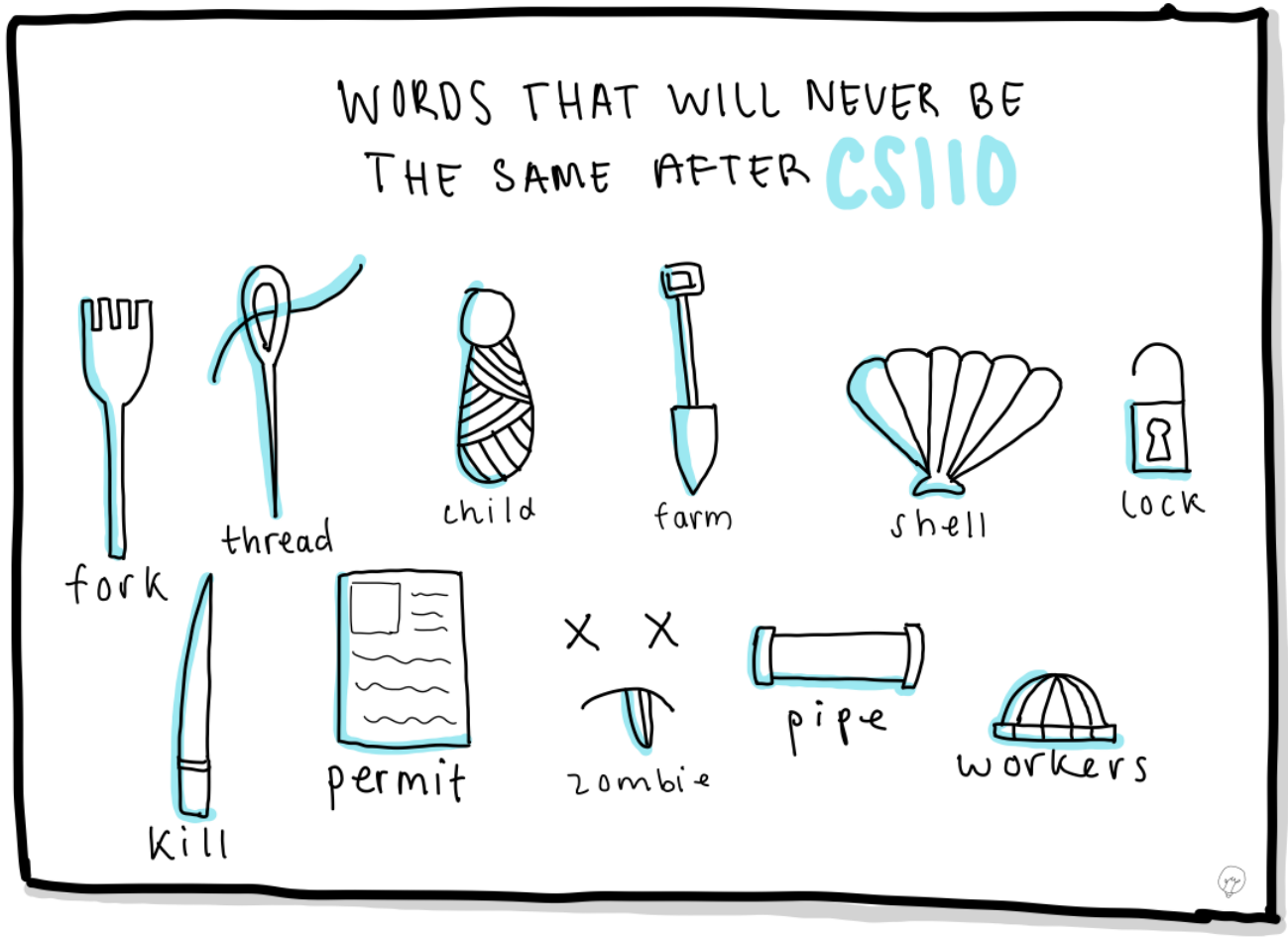


Illustration courtesy of Ecy King, CS110 Champion, Spring 2021

PDF of this presentation

Asking Questions

- Feel free to raise your hand at any time with a question
- If you are more comfortable, you can post a question in the Ed forum thread for each day's lecture (optionally anonymously)
- We will monitor the thread throughout the lecture for questions

Visit Ed (or access via Canvas):

https://edstem.org/us/courses/16701/discussion/

Today's thread:

https://edstem.org/us/courses/16701/discussion/979087

Lecture Plan

- Introduction
- Course Topics Overview
- Course Policies

Lecture Plan

- **Introduction**
- Course Topics Overview
- Course Policies

Guiding Principles For This Quarter

- We are each starting the new year in unique circumstances.
- We are likely not fully recovered or restored from the stresses of the past 2 years and now facing new uncertainties, responsibilities, and emotions.
- We will do everything we can to support you. We have designed the course to the best of our ability to provide flexibility.
- We will constantly evaluate and listen to ensure the class is going as smoothly as possible for everyone.
- Please communicate with us if any personal circumstances or issues arise! We are here to support you.

What is CS110?

CS107 took you behind the scenes:

- how things work inside C++/Python/Java, and how your programs map onto the components of computer systems
- understanding of program behavior and execution

What is CS110?

CS107 took you behind the scenes:

- how things work inside C++/Python/Java, and how your programs map onto the components of computer systems
- understanding of program behavior and execution

CS110 uses this as a foundation to build complex programs that maximally take advantage of the hardware and operating system software available to us:

- How can we understand the designs and tradeoffs of large systems?
- How can we write software that spans multiple machines?
- How can we write software that runs tasks in parallel on a single machine?

Prerequisites

CS107 or equivalent -

- Each of you should know C and C++ reasonably well so that you can...
  - write moderately complex programs (e.g. pointers, **malloc/realloc/free**, C strings, C++ classes, methods, references, templates, **new/delete**)
  - read and understand portions of large code bases
  - trace memory diagrams and always win!
- Each of you should be fluent with **Unix**, **GDB**, **Valgrind**, and **Make** to the extent they're covered in CS107 or its equivalent.

Prerequisites

CS107 or equivalent -

- Each of you should know C and C++ reasonably well so that you can...
  - write moderately complex programs (e.g. pointers, **malloc/realloc/free**, C strings, C++ classes, methods, references, templates, **new/delete**)
  - read and understand portions of large code bases
  - trace memory diagrams and always win!
- Each of you should be fluent with **Unix**, **GDB**, **Valgrind**, and **Make** to the extent they're covered in CS107 or its equivalent.

The first assignment is meant to give you a sense of the scope of CS110 programs and refresh your memory on relevant prerequisites. If you feel ok about it, you're all set!

What do you hope to achieve this quarter in CS110?

Teaching Team

Nick Troccoli (Instructor)

Jerry Cain (Instructor)

Sophie Decoppet (CA)

Ayelet Drazen (CA)

Jonathan Kula (CA)

Victor Lin (CA)

Swayam Parida (CA)

Joel Ramirez (CA)

Instructors

Nick Troccoli

- Stanford BS/MS (coterm) in CS - systems track undergrad, AI track grad
- Lecturer in CS, teaching CS106X, 107, 110
- Systems has played a key part in my discovery of CS, and CS110 was one of my favorite classes!

Jerry Cain

- Chemistry undergrad MIT, originally chemistry Ph.D. student here, defected to CS
- Senior Lecturer in CS, teaching CS106AX, CS106X, CS107, CS109, and CS110
- Taught CS110 for the first time in Spring 2013, and I absolutely love teaching it!
  - CS110 is still an evolving system—particularly given the upheaval of the last 22 months—but hopefully you don't notice one bit
  - Introduced my own materials since then, and will introduce even more this time
  - Leveraged much of Mendel Rosenblum's CS110 materials from prior offerings
- Started working at Facebook in 2008, still with them in an emeritus role
  - Have grown to understand and appreciate large systems much better as a result of working there
  - Learned web programming, PHP, CSS, JavaScript. Old CS107 student of mine (class of 2004) is my manager

Companion Class: CS110A

- CS110A is an extra 1-unit “Pathfinders” or “ACE” section with additional course support, practice and instruction.
- Meets for an additional weekly section and has additional review sessions
- Section Tues. 11AM-1PM
- Entry by application, due 1/7 at 5PM: click right here
- see the CS110 course homepage for more details: cs110.stanford.edu

Amrita Kaur (CS110A CA)

Companion Class: CS110L

- CS110L is an extra 2-unit course that digs a bit deeper into the CS110 material for those who are interested
- The goal is to offer an approachable and engaging exposure to effective, robust, and secure systems programming.
- Program in Rust
- see the CS110L course homepage for more details: cs110l.stanford.edu

Thea Rossman (CS110L CA)

Lecture Plan

- Introduction
- **Course Topics Overview**
- Course Policies

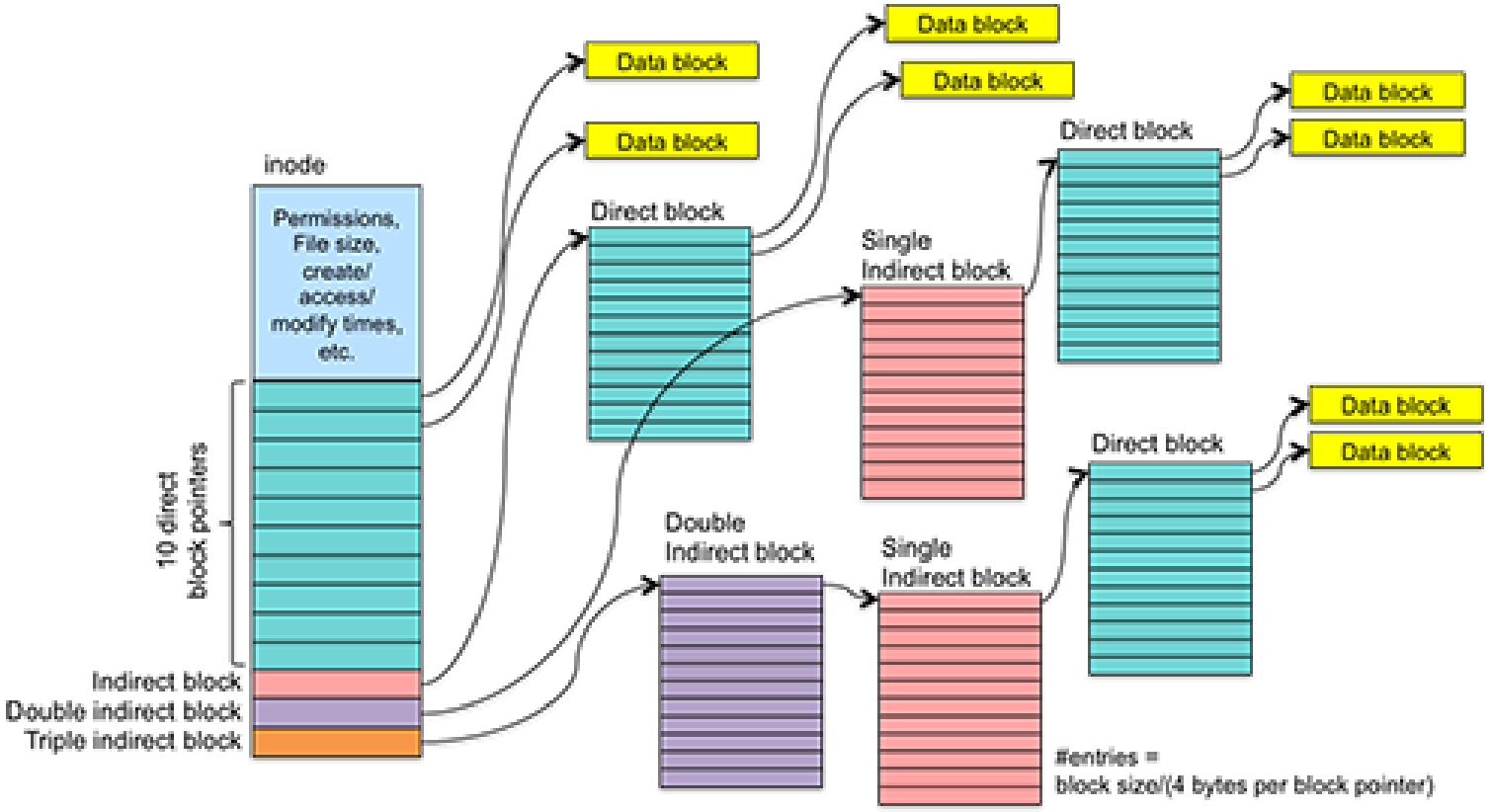
Course Topics Overview

1. **Overview of Linux Filesystems** - *How can we design filesystems to store and manipulate files on disk?*
2. **Multiprocessing and Exceptional Control Flow** - *How can our program create and interact with other programs?*
3. **Threading and Concurrency** - *How can a single instance of our program perform multiple coordinated tasks at the same time?*
4. **Networking and Distributed Computing** - *How can we write programs that communicate over a network with other programs, and tackle large tasks using many machines?*
5. **Additional Topics:** MapReduce, Caching, and Non-Blocking I/O

Full course topic list and calendar: click here

1. Overview of Linux Filesystems

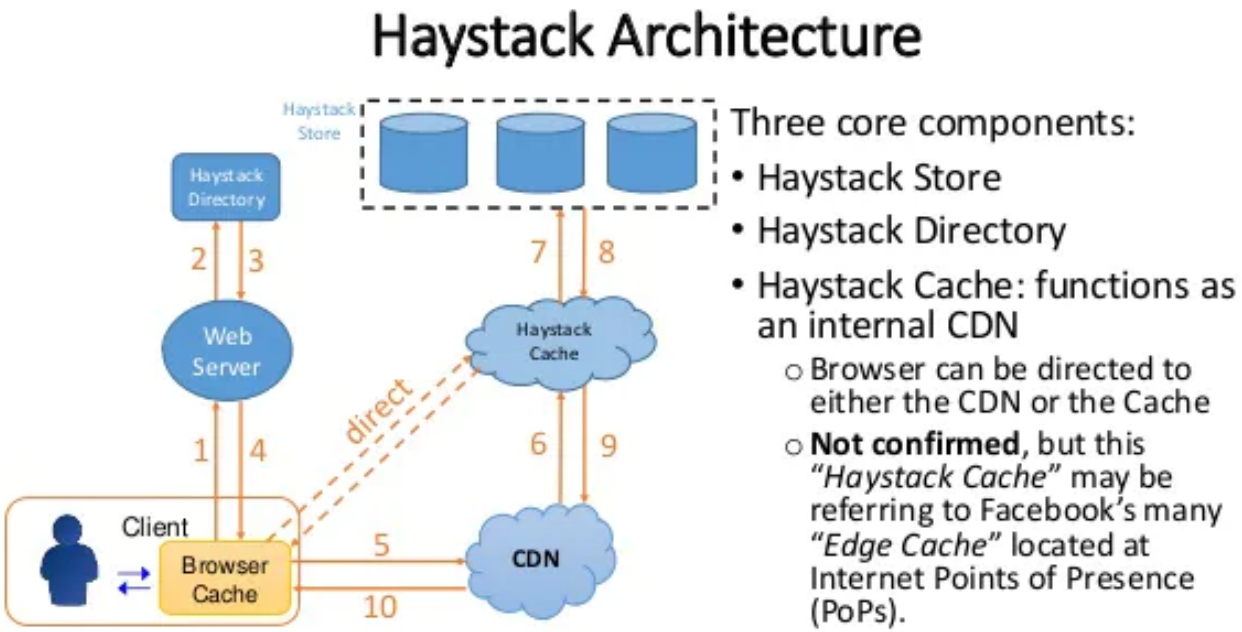
Key Question: *How can we design filesystems to store and manipulate files on disk?*



Unix Filesystem Inode Design [source]

1. Overview of Linux Filesystems

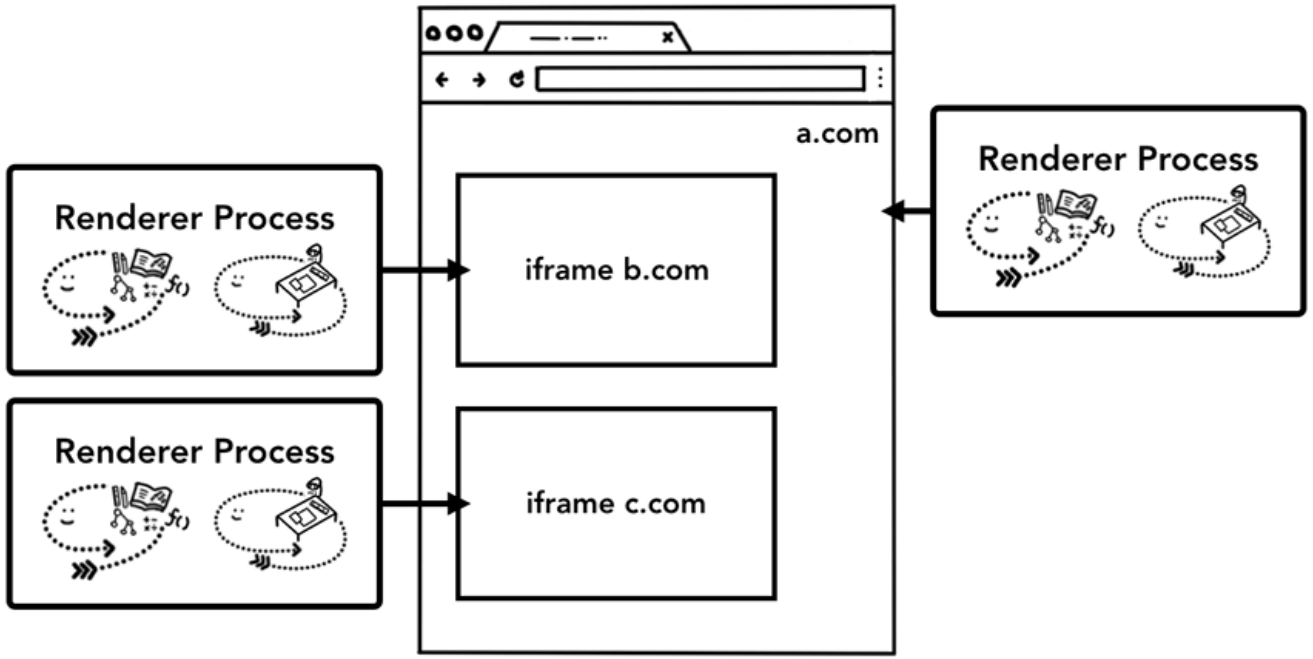
Key Question: *How can we design filesystems to store and manipulate files on disk?*



Facebook's First Generation Haystack [source]

2. Multiprocessing and Exceptional Control Flow

Key Question: *How can our program create and interact with other programs?*

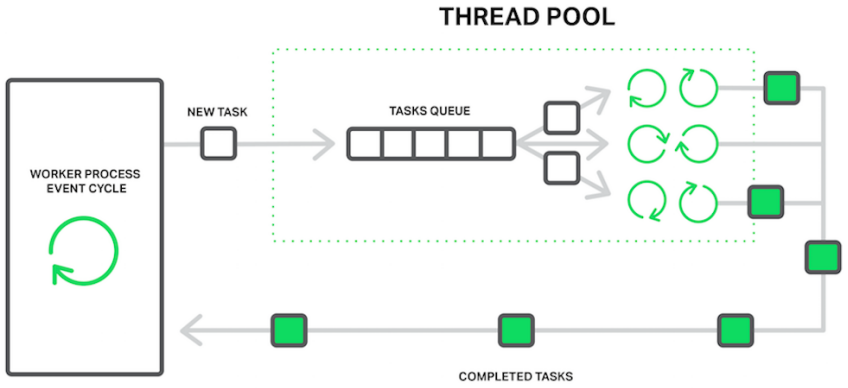


Chrome Site Isolation [source]

3. Threading and Concurrency

Key Question: *How can a single instance of our program perform multiple coordinated tasks at the same time?*

- multi-core processors in many devices to execute tasks in parallel
- challenging to coordinate different tasks!

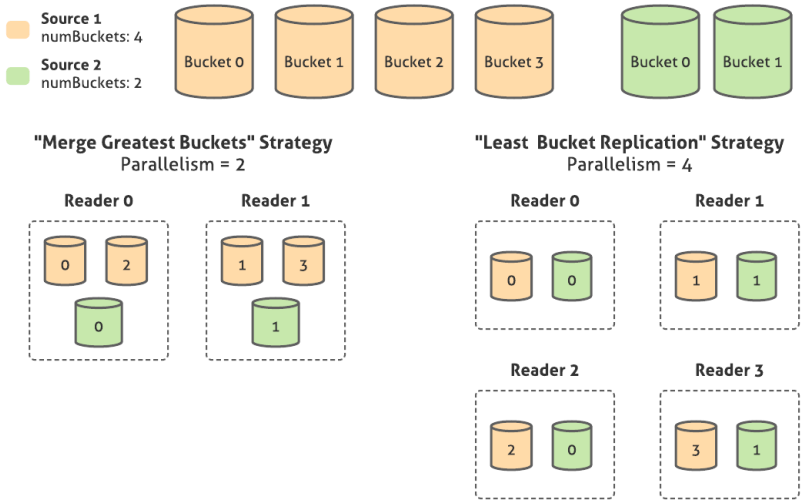


Nginx and Thread Pools [source]

4. Networking and Distributed Computing

Key Question: *How can we write programs that communicate over a network with other programs, and tackle large tasks using many machines?*

- Networking on Linux similar to file reading/writing
- Learn how our devices send/receive data (e.g. web browsers)



Dataflow for Spotify Wrapped [source]

Lecture Plan

- Introduction
- Course Topics Overview
- **Course Policies**

Course Resources/Tools

- **Course Website:** cs110.stanford.edu
- **Canvas** for lecture videos and Zoom links
- **Gradescope** for concept checks
- **EdStem** for discussion forum
- **Zoom** for lecture/section during the first two weeks
- **Slack** for fostering community and getting to know each other

Course Information and Calendar

click here: course information

click here: calendar

Textbooks

Second half of CS107 Textbook: *Computer Systems: A Programmer's Perspective* by Bryant & O'Hallaron, 3rd Edition

- Can purchase full copy, or Stanford Bookstore custom edition with just CS110 chapters

Principles of Computer System Design: An Introduction by Jerome H. Saltzer and M. Frans Kaashoek

- Free Stanford online access here (also linked on course homepage)
- Fewer readings from this book vs. first one
- You're welcome to buy a physical copy if you'd like

Course Structure

- **Lectures:** understand concepts, see demos
- **Lab Sections**: practice concepts, discuss with peers - ***great preview of homework!***
- **Assignments**: build programming skills, synthesize lecture/lab content

Read our full course policies document:

course information

Course Grading

Per-Lecture Concept Checks: 5%

Discussion Section Participation: 5% (or less)

Assignments: 54% (9% x 6)

Assessments: 36% (10% + 10% + 16%)

Read our full course policies document:

course information

Lectures and Concept Checks (5%)

- Lectures are M/W/F 11AM-12PM PDT and are recorded for later viewing
  - slides posted on course website, lecture code available on myth machines
- We *highly encourage* you to attend live to participate in activities and ask questions
- By 5PM each lecture day, we post a short collection of questions on Gradescope about that lecture's material ("concept checks")
  - unlimited submissions, generally written to make it clear when answers are correct
  - concept check due by next lecture
  - late submissions accepted through end of finals week for 85% of the points
  - each concept check weighted equally

Discussion Section Participation (<= 5%)

- Weekly 80-minute sections/labs led by a CA, starting *next week*, offered at various times
- Hands-on practice in small groups with lecture material and course concepts
- Graded on attendance and participation
- Lab preference submissions open **Tuesday 1/4** and are **not first- come first-serve**. You may submit your preferences anytime until **Saturday 1/8**. Sign up on the course website.
- Your total discussion section grade is 100%. However, every time you miss a section, your discussion section grade weight is reduced by 1% and your final (3rd) assessment grade weight is increased by 1%.

Question Break!

What questions do you have about the topics so far? (course structure/goals, textbooks, lectures / concept checks or discussion sections)

Assignments (54%)

- 6 programming assignments, each weighted 9%, completed individually using Unix command line tools / the Myth machines
- Graded on functionality (automated tests - 85%) and style (TA code review - 15%)
- Late policy - if you submit...
  - up to 24 hours later: 95% cap
  - 24-48 hours later: 90% cap
  - more than 48 hours later: 85% cap - but you must notify us
- If you get less than an 85% functionality score, you may resubmit, capped at 85%
- Extensions for exceptional circumstances must be approved by Nick and Jerry. Please communicate with us! We are here to accommodate you as much as possible.
- **You must get at least 70% functionality on each assignment to pass the class.**

Assessments (36%)

- 3 assessments this quarter at the end of weeks 3, 6, and during our final exam slot
- First two assessments are take-home (given 48-hour window when you can take the assessment for 3 hours), third assessment is in-person
- First two assessments 10%; last assessment 16%
- First two assessments open-book, final assessment is closed-book
- If you have testing accommodations, please let us know as soon as possible

Course Grading

Per-Lecture Concept Checks: 5%

Discussion Section Participation: 5% (or less)

Assignments: 54% (9% x 6)

Assessments: 36% (10% + 10% + 16%)

Read our full course policies document:

course information

Course Communication

- Post on the **Discussion Forum**
  - Course material discussions, course policy questions or assignment questions (but don't post assignment code). Course staff monitors the forum and responds.
- Visit **Office Hours with Nick, Jerry and the CAs**
  - Course staff cannot ever look at your assignment code (the exception is for the first assignment, which gets everyone up to speed). This doesn't mean we can't answer code-level questions, but we cannot look at your code to help you debug it.
- Message with other students on **Slack**
  - Coordinate study groups, review for assessments - not meant for course material help from course staff
- **Email** the Course Staff
  - Private matters, not meant for course material help

Honor Code

- Please take the honor code seriously, because the CS Department does
  - Everything you submit for a grade is expected to be original work
  - Provide detailed citations of all sources and collaborations
  - The following are clear no-no's
    - Looking at another student's code
    - Showing another student your code
    - Discussing assignments in such detail that you duplicate a portion of someone else's code in your own program
    - Uploading your code to a public repository (e.g. github) so others can find it - if you'd like to upload your code to a **private** repository, you can do so
- For take-home assessments, you may not actively collaborate with others in person or online. In general, you can only rely on materials that are discoverable by all students via search engines.
- Tutoring is *not* appropriate for help with work that will be submitted for a grade.

Honor Code

- Assignments are checked regularly for similarity with help of software tools.
- If you need help, please contact us and we will help you.
- We do not want you to feel any pressure to violate the Honor Code in order to succeed in this course.
- If you realize that you have made a mistake, you may retract your submission to any assignment at any time, no questions asked.

Question Break!

What questions do you have about the topics so far? (assignments, assessments, course support, or honor code)

Recap

- Introduction
- Course Topics Overview
- Course Policies