Each 6.1800 lecture will come with an outline. You can fill this in during lecture, after lecture, or not at all — it's entirely up to you how you use it. The goal of these outlines is to help you understand the main points that you should be taking away from each lecture. In some cases we will also include examples of things you should be able to do after each lecture.

In the past, these outlines have proved to be an effective tool for studying for the exams. Note that the outlines are ***not exhaustive****; there will be topics and nuances in lecture that aren't captured by the outline.*

Lecture 16: Transactions

- What does it mean for an action to be atomic?
- What is the benefit of atomicity?

We're going to attempt to make some bank-transfer code atomic in this lecture. It's important that you understand the problem with each of our attempts.

- Initial bank transfer code
    - What's the problem with a crash between the two main lines of this code?
- Approach #1 (uses a spreadsheet)
    - Why is it now okay if the system crashes between the same two lines as before?
    - Why is this code still not atomic?
- Approach #2 (shadow copies)
    - Why is a crash during `write_accounts` now okay?
    - Making `rename` atomic:
        - Why is it reasonable to make `rename` atomic (vs. `write_accounts`)?
        - What do we mean by "commit point"?
        - What is the commit point within `rename`?
        - Why is it reasonable for that line of code to be the commit point?
        - What is the point of the recovery process in the context of this code?
    - How well do shadow copies perform?

- What do we mean by isolation?
- Transactions provide atomicity and isolation. What is good about this abstraction?