Model View Controller: MVC

- Model stores and updates state of application
  - Example: calculator, what's the state of a GUI-calculator?
  - When model changes it notifies its views
    - Example: pressing a button on calculator, what happens?
- The controller interprets commands, forwards them appropriately to model (usually not to view)
  - Code for calculator that reacts to button presses
  - Controller isn't always a separate class, often part of GUI-based view in M/VC

Model, View, Controller

- MVC is a fundamental *design pattern*: solution to a problem at a general level, not specific code per se
  - This is a pattern, so there's isn't *"one right way"*
- Model encapsulates state, e.g., documents viewed, game being played
  - For browser: keep list of favorites, documents, …
  - For game: interpret moves, change state, …
  - When model changes, it notifies the view

How do we use a view?

- The view knows about model
  - Construct view with model, pass to view, …
- The model knows about the view
  - Why can't this happen at model construction time?
- Hollywood principle for OO/MVC
  - Don't call us, we'll call you
  - The view calls the model when things happen
  - The model reacts and updates the view, repeat

What about loading files?

- Where are files loaded, model or view?
  - Why is one better? Is one better?
- What about time-consuming operations
  - What if we load a big file, a URL that's blocked
- How do we cope with long-running tasks?
  - Use threads, very hard to do this right.