Today

- MVC
  - Object-Oriented Design Pattern

- Continuation of Calculator Demo
  - Computed Properties, MVC, Laying out the UI to work with different devices
Divide objects in your program into 3 “camps.”
Model = **What** your application is (but not **how** it is displayed)
Controller = How your Model is presented to the user (UI logic)
View = Your Controller's minions
It's all about managing communication between camps.
Controllers can always talk directly to their Model.
Controllers can also talk directly to their View.
The Model and View should never speak to each other.
Can the View speak to its Controller?
Sort of. Communication is “blind” and structured.
The Controller can drop a target on itself.
Then hand out an action to the View.
Then hand out an action to the View.
The **View** sends the **action** when things happen in the UI.
Sometimes the **View** needs to synchronize with the **Controller**.
The **Controller** sets itself as the **View**'s delegate.
The delegate is set via a protocol (i.e. it’s “blind” to class).
Views do not own the data they display.
So, if needed, they have a protocol to acquire it.
Controllers are almost always that data source (not Model!).
Controllers interpret/format Model information for the View.
Can the Model talk directly to the Controller?
No. The **Model** is (should be) UI independent.
So what if the **Model** has information to update or something?
It uses a “radio station”-like broadcast mechanism.
Controllers (or other Model) “tune in” to interesting stuff.
A View might “tune in,” but probably not to a Model’s “station.”
Now combine MVC groups to make complicated programs ...
MVCs working together
MVCs not working together
Demo

Calculator continued ...

- MVC
- struct vs. class (mutating, etc.)
- public versus private API
- more examples of Optional
- Dictionary<KeyType,ValueType>
- enum
- associated values
- switch
- Functions as types
- Closure syntax for defining functions “on the fly”
- UIStackView
- First peek at Autolayout (stick things to the edges)