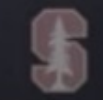
# Shape

- Shape

  Shape is a protocol that inherits from View.

  In other words, all Shapes are also Views.



# Shape

- Shape

  By default, Shapes draw themselves by filling with the current foreground color.
  But we've already seen that this can be changed with `.stroke()` and `.fill()`.
  They return a `View` that draws the Shape in the specified way (by stroking or filling).



# Shape

- Shape

  By default, Shapes draw themselves by filling with the current foreground color.
  But we've already seen that this can be changed with `.stroke()` and `.fill()`.
  They return a `View` that draws the Shape in the specified way (by stroking or filling).

  The arguments to `stroke` and `fill` are pretty interesting.
  In our demo, it looked like the argument to fill was a `Color` (e.g. `Color.white`).
  But that's not quite the case ...



# Shape

- Shape

  By default, Shapes draw themselves by filling with the current foreground color.
  But we've already seen that this can be changed with `.stroke()` and `.fill()`.
  They return a `View` that draws the `Shape` in the specified way (by stroking or filling).

  The arguments to `stroke` and `fill` are pretty interesting.
  In our demo, it looked like the argument to fill was a `Color` (e.g. `Color.white`).
  But that's not quite the case ...

  ```
  func fill<S>(_ whatToFillWith: S) -> some View where S: ShapeStyle
  ```

  This is a generic function (similar to, but different than, a generic type).
  `S` is a don't care (but since there's a `where`, it becomes a "care a little bit").
  `S` can be anything that implements the `ShapeStyle` protocol.
  The `ShapeStyle` protocol turns a `Shape` into a `View` by apply some styling to it.
  Examples of such things: `Color`, `ImagePaint`, `AngularGradient`, `LinearGradient`.



# Shape

- Shape

  But what if you want to create your own Shape?



# Shape

- Shape

  But what if you want to create your own Shape?

  The Shape protocol (by extension) implements View's body var for you.
  But it introduces its own func that you are required to implement …

```
func path(in rect: CGRect) -> Path {
    return a Path
}
```

  In here you will create and return a Path that draws anything you want.
  Path has a ton of functions to support drawing (check out its documentation).
  It can add lines, arcs, bezier curves, etc., together to make a shape.



# Shape

- Shape

  But what if you want to create your own Shape?

  The Shape protocol (by extension) implements View's body var for you.
  But it introduces its own func that you are required to implement …

```
func path(in rect: CGRect) -> Path {
    return a Path
}
```

  In here you will create and return a Path that draws anything you want.
  Path has a ton of functions to support drawing (check out its documentation).
  It can add lines, arcs, bezier curves, etc., together to make a shape.

  This is best shown via demo.
  So we're going to add that "pie" to our CardView (unanimated for now) …

