@ViewBuilder

What exactly is that argument to ZStack, ForEach, LazyVGrid, etc.?

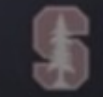


@ViewBuilder

- @ViewBuilder

  Based on a general technology added to Swift to support "list-oriented syntax".

  It's a simple mechanism for supporting a more convenient syntax for lists of Views.



@ViewBuilder

- @ViewBuilder

  Based on a general technology added to Swift to support "list-oriented syntax".
  It's a simple mechanism for supporting a more convenient syntax for lists of Views.

  Developers can apply it to any of their functions that return something that conforms to View.
  If applied, the function still returns something that conforms to View
  But it will do so by interpreting the contents as a list of Views and combines them into one.



@ViewBuilder

- @ViewBuilder

  Based on a general technology added to Swift to support "list-oriented syntax".
  It's a simple mechanism for supporting a more convenient syntax for lists of Views.

  Developers can apply it to any of their functions that return something that conforms to View.
  If applied, the function still returns something that conforms to View
  But it will do so by interpreting the contents as a list of Views and combines them into one.

  That one View that it combines it into might be a TupleView (for two or more Views).
  Or it could be a _ConditionalContent View (when there's an if-else in there).
  Or it could even be EmptyView (if there's nothing at all in there; weird, but allowed).
  And it can be any combination of the above (if's inside other if's, etc.).



@ViewBuilder

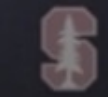
- @ViewBuilder

  Based on a general technology added to Swift to support "list-oriented syntax".
  It's a simple mechanism for supporting a more convenient syntax for lists of Views.

  Developers can apply it to any of their functions that return something that conforms to View.
  If applied, the function still returns something that conforms to View
  But it will do so by interpreting the contents as a list of Views and combines them into one.

  That one View that it combines it into might be a TupleView (for two or more Views).
  Or it could be a _ConditionalContent View (when there's an if-else in there).
  Or it could even be EmptyView (if there's nothing at all in there; weird, but allowed).
  And it can be any combination of the above (if's inside other if's, etc.).

  Note that some of this is not yet fully public API (like _ConditionalContent).
  But we don't actually care what View it creates for us when it combines the Views in the list.
  It's always just some View as far as we're concerned.



@ViewBuilder

- @ViewBuilder

  Any func or read-only computed var can be marked with @ViewBuilder.

  If so marked, the contents of that func or var will be interpreted as a list of Views.

  For example, if we wanted to factor out the Views we use to make the front of a Card ...



@ViewBuilder

- @ViewBuilder

  Any func or read-only computed var can be marked with @ViewBuilder.

  If so marked, the contents of that func or var will be interpreted as a list of Views.

  For example, if we wanted to factor out the Views we use to make the front of a Card …

```
@ViewBuilder
func front(of card: Card) -> some View {
    let shape = RoundedRectangle(cornerRadius: 20)
    shape
    shape.stroke()
    Text(card.content)
}
```

  And it would be legal to put simple if-else's to control which Views are included in the list.
  (But this is just the front of our card, so we don't need any ifs.)

  The above would return a TupleView<RoundedRectangle,RoundedRectangle,Text>.



@ViewBuilder

- @ViewBuilder

  You can also use `@ViewBuilder` to mark a parameter of a function or an init.

  That argument's type must be "a function that returns a View".

  ZStack, HStack, VStack, ForEach, LazyVGrid, etc. all do this (their `content:` parameter).



@ViewBuilder

- @ViewBuilder

  Just to reiterate …

  The contents of a @ViewBuilder is just a list of Views.

  It's not arbitrary code.

  if-else (or switch or if let) statements can be used to choose Views to include in the list.

  You can also have local lets.

  No other kinds of code is allowed (at least as of the time of this lecture).

