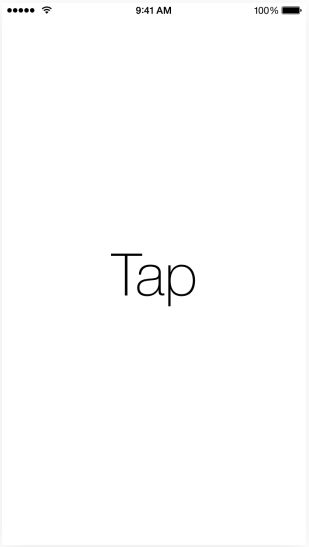
Gesturizer
Lesson 3

Description

Refactor the label-changing code, and apply a `UIView` animation with a closure to achieve a fade-in effect.



Learning Outcomes

- Recognize duplicate code and refactor common operations into a controller helper method.
- Discover the concept of alpha transparency, and plan a means of achieving a visual fade-in effect.
- Describe the concept of closures, and recognize Swift closure syntax.

Vocabulary

refactor	Attributes Inspector	alpha transparency
`UIView`	closure	type annotation
parameter list	return type	`Void`
closure body	`in`	trailing closure

Materials

- **Gesturizer Lesson 3** Xcode project
- **Closures** presentation

Opening

How might we get the label to fade in and then disappear after each gesture?

Agenda

- Discuss the `singleTap:` and `doubleTap:` controller methods, and discuss how both methods duplicate the work of setting the label text and making the label appear.
- Create a new method called `showGestureName:`.

```
func showGestureName(name: String) {
   gestureName.text = name
   gestureName.hidden = false
}
```

- Refactor `singleTap:` and `doubleTap:` to use the new method.

```
@IBAction func singleTap(sender: UITapGestureRecognizer) {
   showGestureName("Tap")
}

@IBAction func doubleTap(sender: UITapGestureRecognizer) {
   showGestureName("Double Tap")
}
```

- Run the app (⌘R), and observe that the functionality remains unchanged.
- Discuss how, to achieve a fade in/out effect, the label should start in an invisible, fully transparent state, slowly become less transparent, and then fade out again.
- Using Interface Builder, select the text label and open the Attributes Inspector (⌥⌘4). Uncheck the *Drawing* > *Hidden* attribute, and set the *Alpha* attribute to **0**.
- Explain the nature of alpha transparency as a decimal number between 0 (transparent) and 1.0 (opaque).
- Using the Xcode Documentation and API Reference (⇧⌘0), explore the `UIView` class reference and the `animateWithDuration:animations:` class method.
- Discuss how the data type of the `animations:` parameter of the `animateWithDuration:animations:` class method describes a closure that expects no parameters and returns nothing.
- Present the concept of closures.
- Discuss how functions are "named closures."
- Add a `makeLabelOpaque` method to the `ViewController` class for changing the alpha transparency of the label.

```
func makeLabelOpaque() -> Void {
   gestureName.alpha = 1.0
}
```

- Discuss how the type of the `makeLabelOpaque` function matches the type of the closure that `animateWithDuration:animations:` expects to receive, but that Swift infers a `Void` return type when a function omits an explicit return type.
- Remove the return type from the `makeLabelOpaque` method.

```
func makeLabelOpaque() {
...
```

- Update the implementation of `showGestureName:` to execute an animation.

```
func showGestureName(name: String) {
   gestureName.text = name
   UIView.animateWithDuration(1.0, animations: makeLabelOpaque)
}
```

- Discuss how the `animations:` argument is the name of the `makeLabelOpaque` function, and not a function call.
- Run the app (⌘R), tap the screen, and observe the label fade into view.
- Explain how the `animateWithDuration:animations:` class method receives a closure containing code that will affect the animatable properties of the view, such as its alpha transparency; and how the method will take care of displaying a smooth, animated transition between the view's initial state, and the change made to the view within the closure.
- Discuss an alternative to passing a function name as a closure argument: using a closure expression.
- Delete the `makeLabelOpaque` method, and update the call to `animateWithDuration:animations:`, using a closure expression.

```
UIView.animateWithDuration(1.0, animations: { () -> Void in
   self.gestureName.alpha = 1.0
})
```

- Explain the closure expression syntax, including the braces, type annotation, and the use of `in` to separate the type annotation from the body of the closure.
- Discuss how the closure expression can be made more succinct by removing the explicit return type.
- Remove the return type from the closure expression.

```
UIView.animateWithDuration(1.0, animations: { () in
   self.gestureName.alpha = 1.0
})
```

- Explain how Swift also infers an empty parameter list when omitted.
- Remove the empty parameter list and the `in` keyword from the closure expression.

```
UIView.animateWithDuration(1.0, animations: {
   self.gestureName.alpha = 1.0
})
```

- Explain that Swift supports a shorthand "trailing closure syntax" when the closure is the last parameter in a parameter list.
- Refactor the call to `animateWithDuration:animations:` with a trailing closure.

```
UIView.animateWithDuration(1.0) { self.gestureName.alpha = 1.0 }
```

- Discuss the benefits of the succinctness of the trailing closure syntax.
- Run the app (⌘R), tap the screen, and observe that the fade-in functionality remains the same.
- Discuss the visible behavior of the interface as it relates to the code in `showGestureName:` to support the comprehension of animation and closures.
- With the app still running, tap or double tap the screen again, and observe how the label no longer fades in for subsequent gestures.

Closing

Why do you think the label stops fading in after the first animation effect?

Modifications And Extensions

- Explore the label's `transform` attribute, and the `CGAffineTransformMakeScale` structure. Implement a transformation effect that makes the label appear to fade into, or out from, the screen.

Resources

The Swift Programming Language: Methods https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Methods.html

UIKit User Interface Catalog: About Views https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/index.html

UIView Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html

The Swift Programming Language: Closures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html