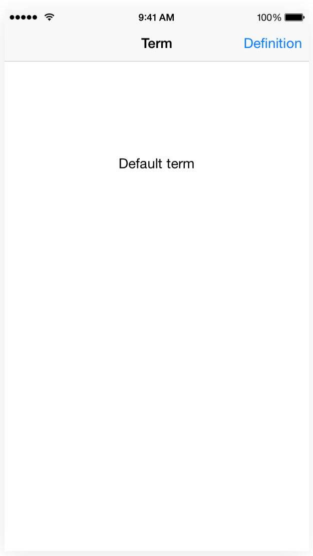
Flashcards
Lesson 2

Description

Add a `Flashcard` model to the project, and update the text label using a model property.



Learning Outcomes

- Practice defining models as Swift classes with properties.
- Describe the purpose of initializers.
- Practice writing initializers to assign initial property values.
- Distinguish between convenience initializers and designated initializers.

Vocabulary

model	class	property
initializer	convenience initializer	designated initializer

Materials

- **Flashcards Lesson 2** Xcode project
- **Initialization** presentation

Opening

What model do you think we need in our app?

Agenda

- Discuss the need for a `Flashcard` model, encapsulating a term and definition.
- Add a new (⌘N) `Flashcard` class to the project.

```
class Flashcard {

}
```

- Discuss how every `Flashcard` object should have a `term` and `definition`.
- Declare `String` properties in the `Flashcard` class for a `term` and `definition`.

```
let term: String
let definition: String
```

- Observe the error notice in the Xcode editor, that indicates the need for an initializer.
- Present the concept of initialization.
- Implement a parameterless initializer in the `Flashcard` class.

```
init() {
   term = "Default term"
   definition = "Default definition."
}
```

- Discuss how a Flashcard object instantiated without parameters will result in a Flashcard object with the `term` **Default term** and the `definition` **Default definition**.
- Add a parameterized initializer to the `Flashcard` class.

```
init(term: String, definition: String) {
   self.term = term
   self.definition = definition
}
```

- Discuss how a `Flashcard` object can now be created with specific `String` values for its `term` and `definition` properties.
- Discuss how the two initializers contain duplicate code that assigns initial values to each property.
- Update the parameterless initializer to invoke the parameterized initializer.

```
init() {
   self.init(term: "Default term", definition: "Default definition.")
}
```

- Observe the error notice in the Xcode editor, indicating the need to declare the parameterless initializer as a convenience initializer.
- Update the parameterless initializer, declaring it as a convenience initializer.

```
convenience init() {
   self.init(term: "Default term", definition: "Default definition.")
}
```

- Explain the designated initializer idiom, describing how "the initializer with the most parameters" is the designated initializer, and is called by other initializers that often supply default values to the designated initializer.
- Discuss how instantiating a `Flashcard` object with either the convenience initializer or the designated initializer creates a `Flashcard` that, at the very least, has valid values for its `term` and `definition` properties.
- Discuss the need for the interface to display a `Flashcard` `term` in the main view of the Term Controller.
- Using Interface Builder and the Assistant Editor (⌥⌘↩), create an outlet for the term label as a `ViewController` property.

```
@IBOutlet weak var termLabel: UILabel!
```

- Update the label in `viewDidLoad` to use a default `Flashcard` object `term`.

```
override func viewDidLoad() {
   super.viewDidLoad()
   let flashcard = Flashcard()
   termLabel.text = flashcard.term
}
```

- Run the app (⌘R), and observe the **Default term** label text.

Closing

If a `Flashcard` had a `color` property, how would you change the initializers? Would you add a third initializer? Which one would be the designated initializer?

Modifications And Extensions

- Delete the convenience initializer and investigate using default parameter values to still enable instantiating `Flashcard` objects without any arguments. Notice how Xcode provides no hint that there are two ways to instantiate a `Flashcard` object. Explain the benefits and drawbacks of declaring an explicit convenience initializer compared to using default parameter values.

- Add a `UIColor` property to the `Flashcard` model, update the initializers, and use the `UIColor` property to add color to the view.
- Replace the `Flashcard` class with a structure, and explain the differences between a class and a structure.

Resources

Cocoa Core Competencies: Model Object https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

The Swift Programming Language: Classes and Structures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

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

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

UILabel Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILabel_Class/